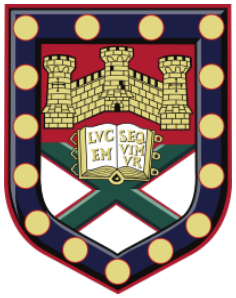
# Uncertainty Quantification for Numerical Models with Two Regions of Solution




**Louise Kimpton**

Supervisors: Peter Challenor

Daniel Williamson

**College of Engineering, Mathematics and Physical Sciences**

**University of Exeter**


Submitted by Louise Kimpton to the University of Exeter as a thesis for the degree of

**Doctor of Philosophy in Mathematics**

**April 2020**



I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

(Signature) .....

# Abstract

Complex numerical models and simulators are essential for representing real life physical systems so that we can make predictions and get a better understanding of the systems themselves. For certain models, the outputs can behave very differently for some input parameters as compared with others, and hence, we end up with distinct bounded regions in the input space. The aim of this thesis is to develop methods for uncertainty quantification for such models.

Emulators act as 'black box' functions to statistically represent the relationships between complex simulator inputs and outputs. It is important not to assume continuity across the output space as there may be discontinuities between the distinct regions. Therefore, it is not possible to use one single Gaussian process emulator (GP) for the entire model. Further, model outputs can take any form and can be either qualitative or quantitative. For example, there may be computer code for a complex model that fails to run for certain input values. In such an example, the output data would correspond to separate binary outcomes of either 'runs' or'fails to run'.

Classification methods can be used to split the input space into separate regions according to their associated outputs. Existing classification methods include logistic regression, which models the probability of being classified into one of two regions. However, to make classification predictions we often draw from an independent Bernoulli distribution (0 represents one region and 1 represents the other), meaning that a distance relationship is lost from the independent draws, and so can result in many misclassifications.

The first section of this thesis presents a new method for classification, where the model outputs are given distinct classifying labels, which are modelled using a

latent Gaussian process. The latent variable is estimated using MCMC sampling, a unique likelihood and distinct prior specifications. The classifier is then verified by calculating a misclassification rate across the input space. By modelling the labels using a latent GP, the major problems associated with logistic regression are avoided. The novel method is applied to a range of examples, including a motivating example which models the hormones associated with the reproductive system in mammals. The two labelled outputs are high and low rates of reproduction.

The remainder of this thesis looks into developing a correlated Bernoulli process to solve the independent drawing problems found when using logistic regression. If simulating chains or fields of 0's and 1's, it is hard to control the 'stickiness' of like symbols. Presented here is a novel approach for a correlated Bernoulli process to create chains of 0's and 1's, for which like symbols cluster together. The structure is used from de Bruijn Graphs - a directed graph, where given a set of symbols, V, and a 'word' length, m, the nodes of the graph consist of all possible sequences of V of length m. De Bruijn Graphs are a generalisation of Markov chains, where the 'word' length controls the number of states that each individual state is dependent on. This increases correlation over a wider area. A de Bruijn process is defined along with run length properties and inference. Ways of expanding this process to higher dimensions are also presented.

# Acknowledgements

I would first like to give a massive thank you to Peter Challenor for all of his help, support and guidance. Thank you for always having confidence in me, even if I had none! I would also like to thank Danny Williamson and Henry Wynn for all of their help and inspiration.

I would next like to thank Gareth, my parents, family (and most importantly Luna) for keeping me going, and supporting me in every way possible (especially with plenty of hugs!). Also thank you to James, Victoria, Evan, Wenzhe and everyone in 601 for always listening if I needed to talk things through or have a rant!

Finally, I would like to thank EPSRC for their studentship funding and allowing me to eat for the past 4 years.

*"Working hard is important. But there is something that matters even more: Believing in yourself."* J.K. Rowling (Harry Potter)

# Table of contents

# List of figures

# List of tables

# Notation

| | |
|---|---|
| $2^i$ | In Chapters 4,5,6 the numerical representation of binary sequences are used. $\sum_{i=1}^{m} k_i \, 2^{i-1}$, where $k_i \in \{0, 1\}$ is each letter in the binary sequence. |
| $\|.\|$ | Norm as a function of Euclidean distance. |
| $< . >$ | Inner product function in space. |
| $*/*'$ | Unknown letter or sequence of 0's and 1's. |
| $B$ | Bayes' factor ratio. |
| $\eta(.)$ | Latent Gaussian process for the labelling function, $\Lambda$. |
| $D$ | Input space for inputs, $\mathbf{x}$. |
| $\delta$ | Correlation length parameter. |
| $f(.)$ | True output function that maps the input $\mathbf{x} \in D$ to the outputs of a model $f(\mathbf{x})$. |
| $G$ | Generating function unless stated otherwise. |
| $i:j$ | Non-directional de Bruijn word of size $m$, where : represents a missing letter. |
| $i\_j$ | Sequences $i$ and $j$ found either end of a run of 1's (or 0's). |
| $l_i$ | Classifying label where $i \in \{1, 2\}$. |
| $\mathcal{L}(\theta; x)$ | Likelihood function. |
| $\lambda$ | Eigenvalues of a matrix unless stated otherwise. |
| $\Lambda(.)$ | Labelling function that maps $\mathbf{x} \in D$ to the labels $\{l_1, l_2\}$. |
| $m$ | Chapter 2: Mean function of the Gaussian process, $\eta$.<br>Chapters 3,4,5,6: de Bruijn word length or size unless stated otherwise. |
| $\mu_4$ | Kurtosis, the fourth central moment. |

| | |
|---|---|
| $n$ | Chapter 2: Dimension of input space. |
| | Chapters 3,4,5,6: de Bruijn run length of 1's (or 0's). |
| $n_i$ | Number of words, $i$, in a given de Bruijn sequence. |
| $p_i^j$ | 1-d directional transition probability from the word $i$ to the word $j$. |
| $p_i^1$ | 2-d directional probability of obtaining a 1 given the word $i$. |
| $p_{i:j}^1$ | 1-d non-directional probability of obtaining a 1 given the word $i:j$. |
| $\pi(\{i\})$ | Marginal probability of obtaining the letter $i$. |
| $\pi(i)$ | Marginal probability of obtaining the word $i$. |
| $\pi(i\_j)$ | Marginal probability of obtaining the sequences $i$ and $j$ either side of a run of 1's (or 0's). |
| $\psi$ | Threshold boundary between two regions. |
| $R_i$ | Regions for classification where $i \in \{1, 2\}$. |
| $s$ | Number of symbols or letters. |
| $S$ | Sequence of 0's and 1's unless stated otherwise. |
| $t$ | Time step. |
| $T$ | Transition matrix of transition probabilities for a Markov chain. |
| $\theta$ | Unknown parameters. |
| $v$ | Chapter 2: Covariance function (or kernel) of the Gaussian process, $\eta$. |
| | Chapters 3,4,5,6: $v \in V$ - Nodes of a graph or set of symbols/letters. |
| $w$ | De Bruijn word (or part of word). |
| $\mathbf{x}$ | Inputs to a model. |

# Chapter 1

# Introduction

It is common to use complex numerical models to represent real life physical systems. Examples include climate science, ecology, economics, biology and astrophysics (Andrianakis et al., 2015; Chang et al., 2014; Vernon et al., 2010). By using numerical models or simulators, we can make predictions and generally gain a better understanding of these complex systems (Sacks et al., 1989). A numerical model is often coded as a computer model which takes in a set of parameters, $\mathbf{x}$, and returns an output, $f(\mathbf{x})$. The output can take a variety of forms including a time series, spatial field or just a single value. The inputs can also take a variety of forms and are typically used to control specific components of the physical processes being modelled. For example, in climate science, we may have a computer model for predicting the temperature of the ocean. The output would simply be the temperature of the water, whilst the inputs could range from spatial location to air temperature and pressure.

Uncertainty quantification refers to a group of methods used to analyse complex numerical models (Craig et al., 2001; Currin et al., 1991; Haylock and O'Hagan, 1996; Kennedy and O'Hagan, 2001). When attempting to describe a real-life system in terms of a computer model, large amounts of uncertainty and error can be introduced, and it is important that we recognise these when analysing the system, or when making inferences. Initially, the computer model is often based on data collected from observations of the system, which itself can produce errors. This can be down to both human and equipment error.

The majority of computer models act as a simplification to the real system when it is either not feasible to model all fine details, or when the simplified version is easier to interpret by the user. It is not always possible to consider every small contribution to an output of the system, however it is important to recognise when these differences between the computer model and the real-life system are likely to occur. Uncertainty is also often introduced in the unknown parameters of the model. Ideally, these parameters will fit exactly with the known data, but this is not always possible due to simplifications and approximations (Hourdin et al., 2017).

The main motivation for this thesis focuses on uncertainty quantification for numerical models where there are two or more output solution regions separated by distinct boundaries. For example, in the form of a tipping point or bifurcation. These differing regions occur when certain parameters in the input space cause the output to the model to behave in an entirely different way to another set of input parameters. Hence, we see the formation of regions in the input space where the outputs for all associated inputs in that region behave in a similar way. Any inputs in a different region will, however, have a distinctly different type of output. An example of this type of model with two output regions is given in Figure 1.1. The inputs to this model are $x \in [0, 25]$, which are split up into two regions according to their associated outputs, $f(x)$. Inputs in the range $x \in [0, 15]$, labelled Region 1, have outputs that follow a sine curve, whilst inputs in the range $x \in (15, 25]$, labelled Region 2, have outputs that follow an exponential curve.

As shown in Figure 1.1, these types of models can induce discontinuities between the regions in the output space, which can create step functions. Hence, it is important that we do not to assume any continuity between the separate solutions. One example of this type of system is in climate science: the Stommel model has a different solution for when the overturning circulation in the ocean is turned off or on (Wunsch, 2005). This thesis focuses on systems where there are exactly two output solutions, with the expectation that the methods developed will extend to systems with more output regions in future work.

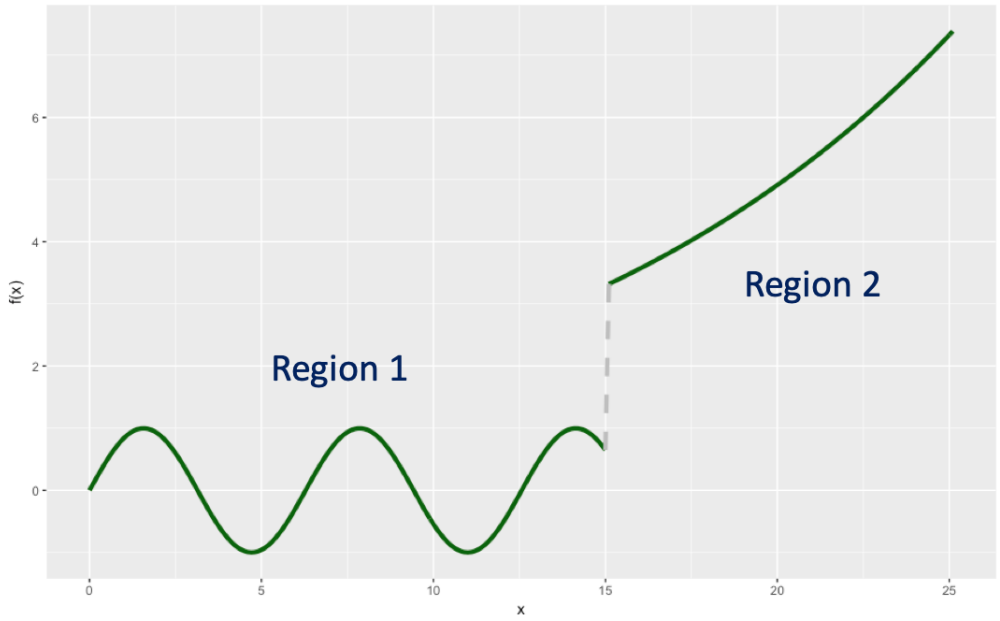



Fig. 1.1 Example of a model with two output regions. Region 1 ($x \in [0, 15]$) follows a sign curve, whilst Region 2 ($x \in (15, 25]$) follows an exponential curve.

Often the relationships between the inputs and outputs of a simulator or model can be represented by a smooth, continuous function which can be modelled as a Gaussian process emulator (GP) (Kennedy and O'Hagan, 2001; Rasmussen and Williams, 2006; Sacks et al., 1989; Santner et al., 2003). Emulators act as a 'black box' model to represent statistically the relationships between the simulator inputs and outputs, providing a deeper understanding of the complex interactions involved in the physical systems, and an approximation of any uncertainty. GPs are a non-parametric approach to regression, such that they find a distribution over the possible functions, $f(\mathbf{x})$, that are consistent with the observed data. A Gaussian process can be seen as a generalisation of a Gaussian distribution over an infinite vector space, and so are fully defined by a mean function, $m(\mathbf{x})$, and a covariance function, $v(\mathbf{x}, \mathbf{x}')$ (Kennedy and O'Hagan, 2001). If a function is distributed as a Gaussian process, then, to work in finite dimensions, only a finite number of samples is required. All marginal, joint and conditional distributions are Normal (Rasmussen and Williams, 2006).

A disadvantage of Gaussian processes is that they assume continuity across the output space (Kennedy and O'Hagan, 2001). Thus, when there are two output regions

which behave very differently to each other, we are highly likely to experience a discontinuity across the border of these regions, and so continuity cannot be assumed. An example of this is shown in Figure 1.2 which shows a one-dimensional problem with a discontinuity in the output space. The true function, $f(x)$, is given in red, where the input range is $x \in [0, 20]$, with $f(x) = -1$ when $x \leq 9$ and $f(x) = 1$ when $x > 9$. Six initial points (orange) are assumed to be known, and a Gaussian process is used to model the function, $f(x)$. The posterior expectation for the GP is shown in blue, along with uncertainty estimates (dotted). For each of the three plots in Figure 1.2, the two middle points are moved such that their distances away from the discontinuity decreases. We can see that when these two points are furthest away from the discontinuity (top plot), the estimate of the boundary between regions is very poor, but the estimates of $f(x)$ away from the boundary are good. As these middle points move closer to the discontinuity, the boundary estimate is improved, but we end up with large fluctuations in the outer areas where the Gaussian process is having to compensate for the abrupt change in scale. Therefore, I can conclude that it is infeasible to model a system with two output solutions with a single Gaussian process.

One approach to this problem in the GP literature is to use non-stationary Gaussian processes. A non-stationary Gaussian process has a covariance structure that varies throughout the input space, where there may be areas of higher variability. This is applicable to models with two output solutions, as the two solutions are assumed to have different output trends, and hence distinct underlying covariance structures. Examples of these include changes to the covariance function (Schmidt and O'Hagan, 2003; Volodina and Williamson, 2020), composite Gaussian processes (Ba and Joseph, 2012) and treed Gaussian processes (Gramacy and Lee, 2009). Treed Gaussian processes partition up the input space to fit different models independently in each region while preserving continuity. Specifically, they divide the input space up by making binary splits on the value based on a single variable so that the boundaries between regions are parallel to coordinate axes. This is an iterative process, such that new partitions are subpartitions of existing partitions. The main problem with

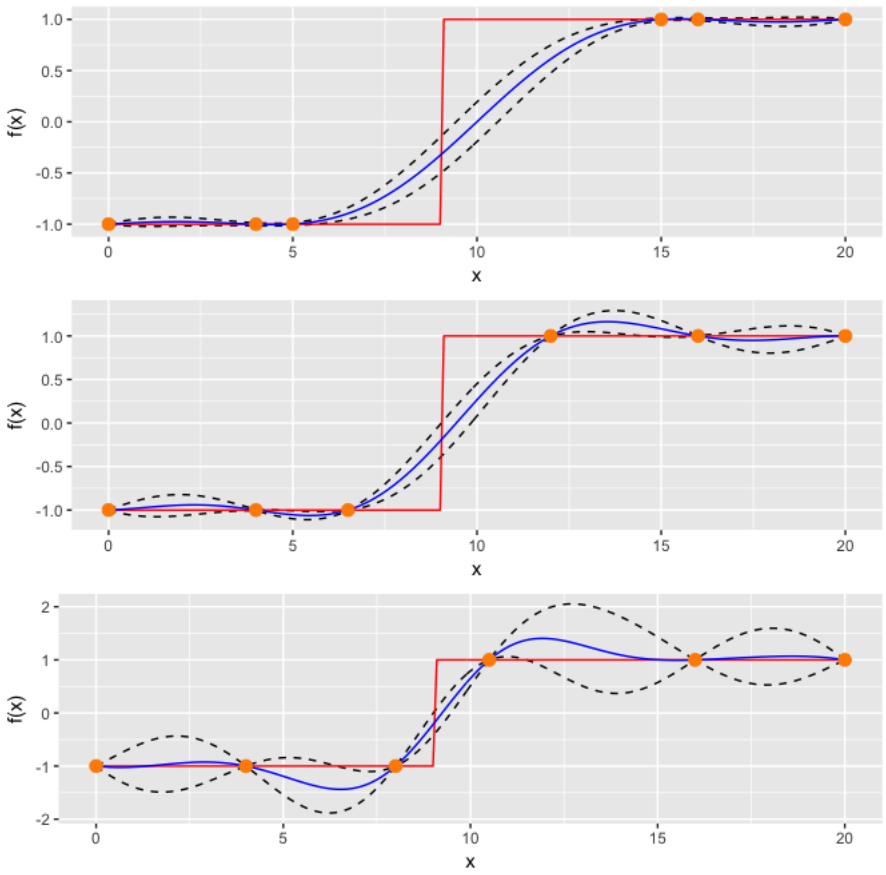



Fig. 1.2 Examples of emulating a step function with a Gaussian process. The true function, $f(x)$, is given in red, where the input range is $x \in [0, 20]$. There are six initial points at varying distances from the discontinuity boundary.

treed Gaussian processes is that partitions are made on straight lines parallel to coordinate axes. This is similar to region partitioning using Voronoi tessellation (Gallier, 2008; Pope et al., 2018) introduced by (Kim et al., 2005), where input space is also partitioned similarly with disadvantages due to straight line partitions. Both of these methods result in a loss of model flexibility, and potential errors when boundaries between output regions are not linear. For the majority of these methods, a quantitative output is required, which restricts the types of models and data that can be used.

To ensure that any method developed here is applicable to a wide range of examples, it is decided that the model outputs can take any form and be either qualitative or quantitative. Possible outputs could be $\{\text{high}, \text{low}\}$, $\{\text{red}, \text{green}\}$ or just $\{0, 1\}$ for the two separate regions. For example, we may have computer code for a complex model that fails to run for certain input values (Edwards et al., 2011).

In such an example, the data would correspond to separate binary outcomes of 'runs' and 'fails to run'.

For these types of data, there is no valid quantitative form for the function, and so there is a need for a method that does not model the entire system using one Gaussian process. An alternative to modelling the system as a whole is to use a method of classification (Rasmussen and Williams, 2006). Classification can be used to split the input space into the separate regions according to their associated outputs. For the computer example above, inputs with the output label 'runs' would be classified as one region (Region 1), and inputs with the output label 'fails to run' would be classified into another region (Region 2). It is sensible to separate the regions as it would be inappropriate to model the system in one region using information gathered from a separate region, where the relationships between inputs and outputs are different. If the numerical model example is such that it has a valid quantitative output, then we may be able to model the regions with separate Gaussian processes once the classification has been completed.

A widely used method for classification is logistic regression (Hilbe, 2009; Kleinbaum and Klein, 1994). Logistic regression is used to model binary variables by modelling the probability of a certain event happening. The binary output variable can take one of two values, 0 or 1, but the inputs to the model can be either continuous or discrete. Improvements have been made by Diggle et al. (1998) to address the concept of using a logit transformation to map the domain of a Gaussian process, $S(x)$, on to the unit interval. The main aim of their paper is to address the assumption of the data being Gaussian distributed, and instead concentrate on situations where the stochastic variation in the data is known to be non-Gaussian. This is an improvement to logistic regression, as the structure of a GP can be incorporated to state levels of uncertainty and spatial variation.

A spatial example is outlined in Diggle et al. (1998) concerning the risk of campylobacter infections, as compared to other infections, in an area of north England. Their data is treated as binomially distributed, relating to whether the infection is either present or not, and is conditional on postcode location and

associated risk. A GP, $S$, is used as a basis of an empirical model used to model the spatial variation through the probability of infections . To use the GP framework, logit transformations are used to map the domain of $S$ to the unit interval. For calculating inference and prediction, the expected number of campylobacter cases $Y_i$ given the GP, $S$, varies spatially only through $S$, and is estimated through MCMC methods. This generates uncertainty in both the systematic and stochastic parts of the model. The data $Y_i$ follow a classical generalised linear model (an example of generalised linear mixed models). The role of the GP here is to explain the residual spatial variation after accounting for all known explanatory variables.

In applying this to the proposed problem, I can consider modelling the separate output regions as Bernoulli trials, where a success translates to being located in one of the specified region or not. Using the logit transformation, we can then model the probability of being classified into one of the two regions.

A similar method is outlined in Chang et al. (2016), where they consider ice sheet models and binary data (0/1 values). The spatial input space is a specified location grid, where the corresponding outputs are binary values which correspond to 1 for when ice is present in a grid cell, and 0 for when no ice is present. The authors' aim is to model the probability of the presence of ice whilst also providing a novel calibration method for computer models whose output is in the form of binary spatial data. Their approach uses a logit transformation in a generalised linear model framework with a latent Gaussian process. By assuming the elements in the model output are conditionally independent given the natural parameters, the likelihood function can be found.

One of the main problems with the methods of Diggle et al. (1998) and Chang et al. (2016), and logistic regression in general, is that the elements in the model output are conditionally independent. For an illustrative example, consider a simple logistic regression that takes inputs $x$ with corresponding binary outputs, $f(x) = 0$ or $f(x) = 1$, where we model the probability $p$, $P(f(x) = 1|x)$. When predicting, it is common to either let $f(x) = 0$ when $p < 0.5$ and $f(x) = 1$ when $p \geq 0.5$, or to take an independent Bernoulli draw from $p$ to give estimates of $f(x) = 0$ or

$f(x) = 1$. An example of this is shown in Figure 1.3. The top plot (red) shows the true function, $f(x)$, where $f(x) = 0$ for $x < 0.5$ and $f(x) = 1$ for $x \geq 0.5$. Through application of a logistic regression, the predicted probability of $f(x) = 1$ for all values of $x \in [0, 1]$ is calculated, and shown in the middle plot (black). We can then take independent Bernoulli draws from the predicted probability to give classification predictions for each value of $x$, given in the bottom plot (blue). Although the model predicts well for lower and higher values of $x$, there is an area around the boundary where this is not the case. When the probability of $f(x) = 1$ is close to 0.5, we become increasingly uncertain of the predicted classification and it is easy for the model to give misclassifications. This is shown by the 'spiky' area in the centre of the plot, which is unlike that of the true function we are looking for.

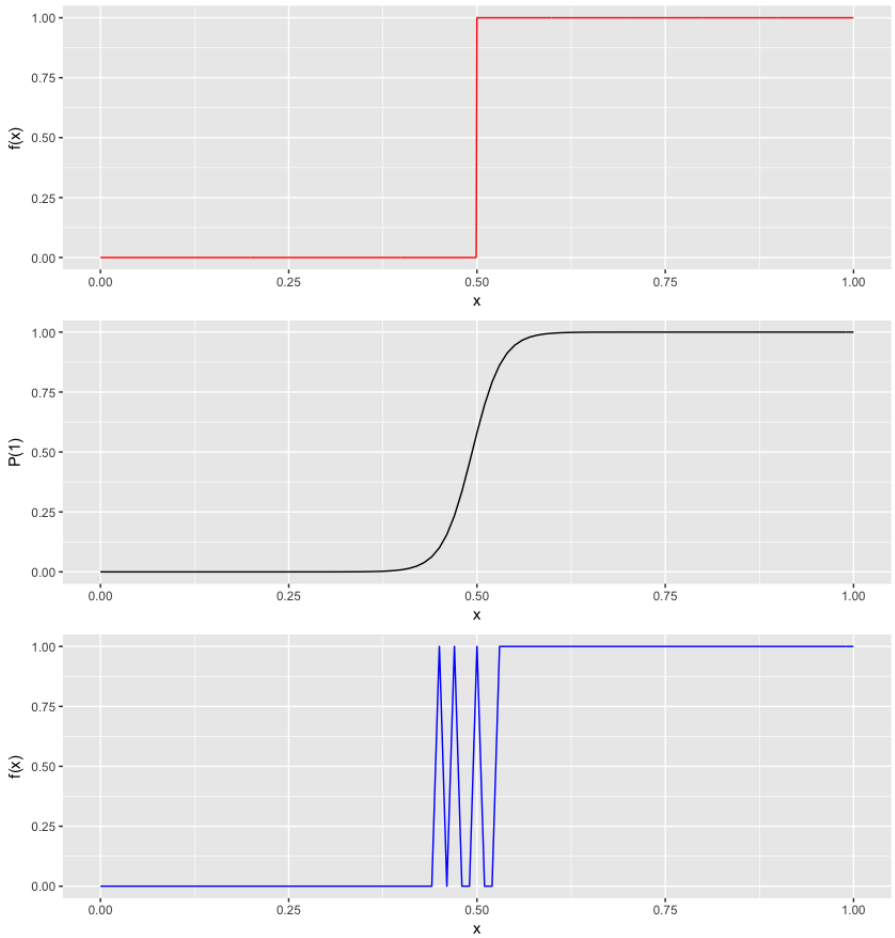



Fig. 1.3 Example of logistic regression. The top plot (red) shows the true function, $f(x) \in \{0, 1\}$. Applying a logistic regression, the predicted probability of $f(x) = 1$ for all values of $x \in [0, 1]$ is given in the middle plot (black). The bottom plot (blue) shows draws from Bernoulli trials with the given probability to give classification predictions for each value of $x$.

The 'spiky' region produced in this plot is due to the lack of distance related correlation in the classification. The logistic regression produces a posterior distribution for the probability of the output taking the value 1, which includes distance information. However, when we sample from the distribution, or use it to make predictions, we draw from an independent Bernoulli distribution. Drawing marginally in this way, instead of from a joint distribution means that any correlation between inputs is lost, causing far more misclassifications to happen. Such drawing is also inconsistent with the correlated logistic regression.

Take a simple example, similar to that in Chang et al. (2016), concerned with classifying areas of ice sheet and ocean. We assume just one spatial input along a line, where we know whether there is any ice sheet present at any of four initial points. The first two points are known to definitely be ice (Region 1) and the following two points are known to be ocean (Region 2). Logically, there should be a much higher chance of finding ice sheet close to where existing points are already known to be ice (Region 1), rather than ocean (Region 2). If we drew points independently close to these known points, then there is still a chance that we may result in a misclassification; it is thus important to include some correlation proportional to distance in our joint sampling.

Suppose we restrict this example with the advanced knowledge that there is only one boundary between ice and ocean in our input space. The change in label can happen anywhere between the two central points, and we assume a hard boundary (i.e. in any realisation any point is either ice or ocean). The input space between each pair of points in the same regions, however, must be classified with the same label as the surrounding points. As we get closer to this boundary, the probability of being classified into the first region becomes close to 50% as we are uncertain of where the exact boundary lies. Hence, the draws from the Bernoulli distribution become equally likely to fall on a 0 or a 1, and so there will be a section (close to the boundary) where the classification may appear random (as seen in Figure 1.3). Our draws are then unrealistic, because we know a hard boundary exists.

Other examples of classification are discussed by Rasmussen and Williams (2006) and Nickisch and Rasmussen (2008). In their works, input data points, $x_i$, are associated with separate class regions with corresponding class labels $y_i \in \{-1, 1\}$. The true system function becomes latent in the model, and is transformed using a sigmoid function, so that the probability of being in one of the classes, $P(y = +1|x)$, can be modelled. The class labels are assumed to be independently distributed Bernoulli random variables. A posterior distribution over the latent values is found, and the predictive class membership probability is obtained to classify new data points. The main disadvantage with the method outlined by Nickisch and Rasmussen (2008) is that part of the posterior distribution is not analytically tractable. This is because the observation likelihood is no longer Gaussian. The remainder of their paper describes different techniques to numerically approximate the posterior distribution for the predictive class membership. These include Laplace Approximation, Expectation Propagation and minimising the Kullback-Leibler divergence. This is also shown by Chan and Dong (2011).

Whilst classical classification exists for sorting data into specified regions, the methods are similar to the methods used by Diggle et al. (1998) and Chang et al. (2016), neglecting any information regarding distance in the input space. This spatial relationship between neighbouring points is valuable and should be incorporated into our classification.

The aim of Chapter 2 is to develop a new method of uncertainty quantification (UQ) classification for computer models that have two distinct labelled regions, and where an output function is not necessarily quantitative across the whole input space. The main aim for this method will be to include a distance correlation between inputs whilst making classification predictions. A latent GP approach with region labelling will be outlined in Chapter 2, along with model validation in the form of a misclassification rate. Examples will be presented for both 1-d and 2-d problems, with details of how we may tackle higher dimensions. Comparisons with existing methods will be discussed, as well as insights into how we may solve the design problem for this type of model. The method will be tested on an application that

models reproduction rates in mammals. Conclusions and future work will be given in Chapter 7.

An alternative to developing an entirely new classification method is to look into multivariate correlated Bernoulli distributions. The main disadvantage with the methods proposed by Diggle et al. (1998) and Chang et al. (2016) is the drawing from an independent Bernoulli distribution. If we were to use these methods for the proposed problem, we would model the probability of being classified into one of two regions, and then draw from this to obtain 0/1 classifications. Drawing a 0 would represent one region, and drawing a 1 would represent the other. As we saw in Figure 1.3, when the probability of being classified into one region approaches 0.5, this can result in many misclassifications because we are ignoring the distance correlation by drawing independently. To solve this problem, I will consider drawing jointly across the whole input space using a multivariate Bernoulli process.

Teugels (1990) introduces both multivariate Bernoulli and Binomial distributions. A sequence of Bernoulli random variables, $\{X_1, X_2, ..., X_n\}$, is considered where, $P\{X_i = 0\} = q_i$ and $P\{X_i = 1\} = p_i$ for $0 < p_i = 1 - q_i < 1$ are the marginal probabilities of obtaining either a 0 or a 1. The author focuses on finding a representation for the multivariate Bernoulli distribution,

$$p_{k_1,k_2,...,k_n} = P\{X_1 = k_1, X_2 = k_2, ..., X_n = k_n\} \ ,$$

assuming that $k_i \in \{0, 1\}$.

The multivariate Bernoulli distribution therefore gives the joint probability of each possible set of zeros and ones (successes and failures) for a given number of individual Bernoulli trials. For example, if $n = 2$, then multivariate Bernoulli would be set out as follows:

$$\begin{aligned} p_{00} =& P\{X_1 = 0, X_2 = 0\} \ , \\ p_{10} =& P\{X_1 = 1, X_2 = 0\} \ , \\ p_{01} =& P\{X_1 = 0, X_2 = 1\} \ , \\ p_{11} =& P\{X_1 = 1, X_2 = 1\} \ , \end{aligned}$$

where, $p_{00}+p_{10}+p_{01}+p_{11} = 1$ . This distribution can be expressed by three parameters, $E[X_1] = P(X_1 = 1)$, $E[X_2] = P(X_2 = 1)$, and $E\left[(X_1 - E[X_1])\,(X_2 - E[X_2])\right]$ which are the central moments of the distribution.

The author continues to consider all cases, expanding to sequences of 0's and 1's of length $n$. For a sequence of length $n$, a probability is stated for each possible combination of 0's and 1's, which makes up the distribution. Therefore, the method requires $2^n$ probabilities to be calculated which are dependent on $2^n - 1$ parameters (either ordinary or central moments). These parameters consist of $E[X_i] = P(X_i = 1)$, for each variable, $X_i$ and $E\left[(X_1 - E[X_1])\,(X_2 - E[X_2])\right], ..., E[(X_1 - E[X_1]) .... (X_n - E[X_n])]$ for every possible combination of variables, $X_1, ..., X_n$.

Although the work by Teugels (1990) successfully defines a multivariate Bernoulli distribution for our classification problem, there are vital improvements to be made. The main problem is the large number of parameters required for even fairly short sequences of 0's and 1's. This would produce large combinatorial problems for sequences even as short as $n = 10$. Hence, it is vital that I either find a way of reducing the number of parameters or develop an alternative method that is less computationally challenging. It may also be difficult to expand this method to higher dimensions, where it is highly likely that the number of parameters will become far too large to cope with.

Secondly, it would be ideal if we could control the amount of correlation that is spread across the 0's and 1's. The aim of this work would be to develop a correlated Bernoulli process so that we could produce sequences of 0's and 1's where correlation is a function of distance. This is similar to the correlation function in Gaussian processes. Enforcing this would mean that points that are closer together are more likely to have the same symbol (0 or 1) and so groups of 0's and 1's would be more likely to appear.

Therefore, Chapters 3, 4, 5 and 6 focus on developing a correlated Bernoulli process. I focus on developing a new method, only taking inspiration from the work by Teugels (1990), using the structures from a concept known as de Bruijn graphs, which are described in Chapter 3. Chapter 3 is also where I will show how we can

use de Bruijn graphs to develop a correlated Bernoulli process, which I will define as a de Bruijn process. Examples are given and further work including properties and inference are detailed in Chapter 4. Chapter 5 attempts to expand the de Bruijn process to two and higher dimensions, whilst Chapter 6 moves towards developing a non-directional de Bruijn process. Conclusions and future work are given in Chapter 7.

# Chapter 2

# Latent Gaussian Processes for Labelled Outputs

## 2.1 Introduction

In Chapter 1, I demonstrated the need for a classification method for complex numerical models with two distinct labelled output solutions. The disadvantages of current methods, such as logistic regression (Chang et al., 2016; Diggle et al., 1998), make it clear that we need to take account of the correlation distance in the input space.

To impose a distance measure on the inputs when classifying, I insist that the input space is a vector space (Strang, 2006). A useful tool which satisfies this constraint is a Gaussian process emulator (O'Hagan, 2006). Gaussian processes enable us to state some levels of uncertainty to any estimates or predictions, which is crucial when working with complex simulators. It is also the case that certain applications do not have quantitative simulator or model function outputs. For example, we may have outputs $\{\text{fail}, \text{not fail}\}$, $\{\text{red}, \text{green}\}$ or just $\{0, 1\}$ for our two separate labelled regions. Therefore, I aim to ensure that the method I will develop here can also cope with these types of problems.

In this Chapter, I develop a new method that encapsulates the ideas of both general classification and uncertainty quantification, which is applicable to a wide range of applications, and includes distance correlation between model inputs. I

will include ideas from Nickisch and Rasmussen (2008) and Chang et al. (2016) to model the regions using class labels in latent space with a Gaussian process, and use Metropolis Hastings and Gibbs sampling Bayesian methods (Brooks et al., 2012) for model estimation. Due to the unique set up of the problem, the likelihood and prior specifications for the Gaussian process are important, and are therefore discussed in detail. As a method of model verification, I perform a leave-one-out cross validation (Rasmussen and Williams, 2006) to calculate a misclassification rate for the input space. A possible method for the design problem based on both trying to improve the classification estimation and providing a space filling design is discussed.

A motivational example has been supplied by Voliotis et al. (2018), the subject of which is the reproductive system in mammals. Their model has two dimensions, consisting of a set of coupled ordinary differential equations describing the quantities of certain hormones linked to the causes of high and low rates of reproduction. Thus, this system has two distinct labelled output solutions (high and low rates). Being able to model the system, and locate the areas of input space associated with low and high rates of reproduction means that not only can we aid predictions on the reproduction rate, but we can also have a better understanding of the specific input parameters that are associated with high rates of reproduction.

In Section 2.2 I give an outline of my method, including a brief overview of Gaussian processes, a simple 1 dimensional example and alternative methods. In Section 2.3, I then discuss my approach to model validation. In Section 2.4, I explain some of the prior choices that are vital to the proposed method. Section 2.5 expands the method to a 2 dimensional example, followed by some comparisons with existing methods in Section 2.6. I then make an attempt at the design problem in Section 2.7, and apply it to a more complicated example in Section 2.8. The motivational example is outlined in Section 2.9. Finally in Section 2.10, I finish with a discussion and overview.

## 2.2 Methodology

### 2.2.1 Latent Model

Let $\mathcal{X}$ be a normed vector space with norm $\|.\|$ (Strang, 2006). Further, let $\mathbf{x} = \mathbf{x}_1, \ldots, \mathbf{x}_n \in D \in \mathcal{X}$ be inputs to a model in $p$ dimensions that lie within $\mathcal{X}$. The input space, $D$, is partitioned into 2 regions, $R_1$ and $R_2$, such that $R_1 \cup R_2 = D$ and $R_1 \cap R_2 = \emptyset$.

The function, $f(.)$, that maps the inputs, $\mathbf{x} \in D$, to the outputs of the model, $f(\mathbf{x})$, may lie in real (or complex) space, but may also be qualitative (e.g. fail/not fail) or simply take values $f(\mathbf{x}) \in \{0, 1\}$. To ensure generalisability of the method, I define the function, $\Lambda(.)$, which assigns a class labelling to each of the input data points, $\mathbf{x}_1, \ldots, \mathbf{x}_n$ as follows:

$$\begin{aligned} \Lambda : D \longmapsto \{l_1, l_2\}; \;\; \Lambda(\mathbf{x}) = l_1 \quad \forall \;\; \mathbf{x} \in R_1, \\ \Lambda(\mathbf{x}) = l_2 \quad \forall \;\; \mathbf{x} \in R_2. \end{aligned} \tag{2.1}$$

To see the applications of this, consider the example of a computer model that only runs to completion for certain inputs, $\mathbf{x}$. The simulator output is labelled, $f(\mathbf{x}) \in \{\text{fail}, \text{not fail}\}$. The inputs that lead to a failed run will lie in $R_1$ and be given label $l_1$ (fail). Then, all inputs that run to completion will lie in $R_2$ and be given label $l_2$ (not fail). By modelling $\Lambda(.)$, instead of the output function, $f(.)$, we are now living in latent space, where the latent variable is a quantity that is not observed directly, but rather inferred from the region observations. For each input point, $\mathbf{x}$, there is a class label, $\Lambda(\mathbf{x})$ and a separate function output, $f(\mathbf{x})$. By modelling $\Lambda(\mathbf{x})$, the classification can be modelled regardless of the form of the output values, $f(\mathbf{x})$.

$\Lambda(\mathbf{x})$ is modelled using a latent Gaussian process (GP), $\eta(\mathbf{x})$, so that:

$$\Lambda(\mathbf{x})|\eta(\mathbf{x}) = \begin{cases} l_1 \quad \forall \;\; \mathbf{x} \in D \;\; : \;\; \eta(\mathbf{x}) < 0 \\ \\ l_2 \quad \forall \;\; \mathbf{x} \in D \;\; : \;\; \eta(\mathbf{x}) \geq 0, \end{cases} \tag{2.2}$$

and $\eta(\mathbf{x}) \sim GP(m(\mathbf{x}), v(\mathbf{x}, \mathbf{x}'))$. Note that $f(\mathbf{x})$ does not enter the model.

A Gaussian process (Haylock and O'Hagan, 1996; O'Hagan, 2006; Sacks et al., 1989; Santner et al., 2003) is a generalisation to infinite dimensions of the normal distribution that defines a distribution over functions (Kennedy and O'Hagan, 2001). Any finite collection of random variables from a Gaussian process has a multivariate Normal distribution. GPs are fully defined by their mean function, $m(.)$, and covariance function (or kernel), $v(.,.)$, (Rasmussen and Williams, 2006) where:

$$m : D \longmapsto \mathbb{R}; \;\; m(\mathbf{x}) = \mathbb{E}[\eta(\mathbf{x})],$$
$$v : D \times D \longmapsto \mathbb{R}; \;\; v(\mathbf{x}, \mathbf{x}') = \mathrm{Cov}[\eta(\mathbf{x}), \eta(\mathbf{x}')].$$

The mean function allows us to input any prior beliefs about the form of $\eta$. Here I will only consider Gaussian processes with linear prior mean functions, specified in the form: $\mathbb{E}[\eta(\mathbf{x})|\boldsymbol{\beta}] = \mathbf{h}(\mathbf{x})^T\boldsymbol{\beta}$, where $\mathbf{h}(\mathbf{x})$ is a vector of basis functions of $\mathbf{x}$, and $\boldsymbol{\beta}$ is a vector comprising of unknown coefficients. The choice of basis function is generally based on expert information about the simulator, or on the behaviours of the actual system. It is assumed that the latent Gaussian process is stationary such that $\mathrm{Cov}[\eta(\mathbf{x}), \eta(\mathbf{x}')]$ is a function of distance, $\|\mathbf{x} - \mathbf{x}'\|$, where $\|\mathbf{x}\| = \langle \mathbf{x}, \mathbf{x}' \rangle$, with $\langle \mathbf{x}, \mathbf{x}' \rangle$, is an inner product in space. The covariance function is written as $\sigma^2 c(\mathbf{x}, \mathbf{x}')$, where $\sigma^2$ is the process variance and $c$ is a known correlation function of $\|\mathbf{x} - \mathbf{x}'\|$. A common choice of correlation function is the squared exponential; $c(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{\delta}\right\}$, where $\delta$ is the correlation length parameter controlling the smoothness of the process (how much it can be perturbed as the inputs are varied). The covariance function ensures that the classification is dependent on the distance between the inputs, $\mathbf{x}$. Other common choices of covariance function include powered exponential and Matern (Paciorek and Schervish, 2006; Rasmussen and Williams, 2006).

To update the prior Gaussian process with data $\Lambda(\mathbf{x}) = \Lambda(x_1), ..., \Lambda(x_n)$, we require the likelihood, $P(\Lambda(\mathbf{x})|\eta(\mathbf{x}))$, where:

$$\begin{aligned} P(\Lambda(\mathbf{x})|\eta(\mathbf{x})) &= P\left(\eta(x_1) < 0, \eta(x_2) < 0, \ldots, \eta(x_j) < 0, \eta(x_{j+1}) > 0, \ldots, \eta(x_n) > 0\right) \\ &= \int_{-\infty}^{0} \cdots \int_{-\infty}^{0} \int_{0}^{\infty} \cdots \int_{0}^{\infty} \phi(\eta(x_1), \eta(x_2), \ldots, \eta(x_j), \eta(x_{j+1}), \ldots, \\ &\qquad \eta(x_n))\ dx_1 dx_2 \ldots dx_n. \end{aligned}$$

Without loss of generality, the first $j$ points are labelled $l_1$, and the remaining points are labelled $l_2$. The main difference between the likelihood of an ordinary Gaussian process and the latent one here is the use of a joint cumulative distribution function instead of the probability density. It is also crucial to note that the likelihood used here is a joint distribution (rather than a product of marginals) as the data are not independent. By choosing the likelihood as so, we put no constraints on the specific values of the generated Gaussian process, just their signs. The form of this likelihood is analogous to the Gaussian process fitting seen in Gosling (2005) and in Gosling et al. (2007). I sample from the posterior, $P(\eta(\mathbf{x})|\Lambda(\mathbf{x}), \boldsymbol{\beta}, \sigma^2, \delta)$, using an MCMC algorithm, which is discussed below. Note that by sampling from a joint distribution, we are are able to classify whole sets of points in the input space simultaneously, as well as individual points.

In order to classify a set of new points, $\mathbf{x}^* = x_1^*, .., x_m^*$, we require joint samples from the posterior predictive distribution:

$$P(\eta(\mathbf{x}^*)|\Lambda(\mathbf{x})) = \int\int P(\eta(\mathbf{x}^*)|\eta(\mathbf{x}), \theta) P(\eta(\mathbf{x})|\theta, \Lambda(\mathbf{x})) P(\theta|\Lambda(\mathbf{x}))\ d\eta(\mathbf{x}) d\theta,$$

where, $\theta = (\boldsymbol{\beta}, \sigma^2, \delta)$.

We can obtain a set of samples from $P(\theta|\Lambda(\mathbf{x}))$ using Metropolis Hastings MCMC (Brooks et al., 2012; Chib and Greenberg, 1995; Gelman et al., 2013). Markov chain Monte Carlo (MCMC) methods refer to a collection of algorithms that are designed to approximate random samples from probability distributions that would cause problems if we were to try sampling directly. The general Metropolis Hastings

algorithm works by iteratively constructing a Markov chain such that its stationary distribution, $p$, is the distribution of interest. As such, the distribution of the next sample is dependent only on the current sample value. A likelihood function is used which must be proportional to the distribution we want to sample from. As the chain of samples gets larger, the distribution of the values becomes increasingly close to estimating the desired distribution exactly. At each time $t$, the next state of the Markov chain, $\theta_{t+1}$, is found by sampling a candidate value, $\theta^*$, from a proposal distribution $q(.|\theta_t)$. The candidate is then accepted with probability $a(\theta^*|\theta) = \min(1, A)$, where:

$$A = \frac{p(\theta^*)q(\theta|\theta^*)}{p(\theta)q(\theta^*|\theta)}.$$

If the candidate is accepted, then the chain moves to the new state, $\theta_{t+1} = \theta^*$. If the candidate is rejected, then the chain still advances to the next step, but its value is exactly the value of the previous state, $\theta_{t+1} = \theta_t$. This process iterates a set number of times until the stationary distribution of the Markov chain is exactly the unknown distribution, $p$.

Given a sample from $P(\eta(\mathbf{x})|\theta, \Lambda(\mathbf{x}))$, we can then easily sample from:

$$\begin{aligned}
\eta(\mathbf{x}^*)|\eta(\mathbf{x}), \theta &\sim \mathcal{MVN}\left(m^*(\mathbf{x}^*), v^*(\mathbf{x}^*, \mathbf{x}^*)\right), \\
m^*(\mathbf{x}^*) &= m(\mathbf{x}^*) + v(\mathbf{x}^*, \mathbf{x})v(\mathbf{x}, \mathbf{x})^{-1}\left(\eta(\mathbf{x}) - m(\mathbf{x})\right), \\
c^*(\mathbf{x}^*, \mathbf{x}^*) &= c(\mathbf{x}^*, \mathbf{x}^*) - c(\mathbf{x}^*, \mathbf{x})c(\mathbf{x}, \mathbf{x})^{-1}c(\mathbf{x}, \mathbf{x}^*).
\end{aligned}$$

Sampling from $P(\eta(\mathbf{x})|\theta, \Lambda(\mathbf{x}))$ is not so straight forward. We could perform a rejection sample, similar to that seen in the ABC (Approximate Bayesian Computation) literature (Turner and Van Zandt, 2012; Wilkinson, 2008), but such methods are extremely inefficient for even modest ensemble size $n$. Rejection sampling requires us to sample many latent Gaussian processes from a multivariate Normal distribution, where the sign of every $\eta(x_i)$ must agree with the respective model labels, $\Lambda(x_i)$, simultaneously. If only one point does not agree with the corresponding label, then we still must reject the whole sample, making this method very inefficient. Therefore, we must include an ABC step (ABC-MCMC) in estimating the GP. The likelihood down-weights the chance for a wrong draw to occur, but there will still be cases

where the GP draws do not match with the negative/positive labelling. An ABC step at the end checks these draws, and removes any that shouldn't be there.

My solution is to use a Gibbs sampler (Brooks et al., 2012; Gilks et al., 1996), making use of the full conditional distributions by going through each variable in turn to sample from its conditional distribution, whilst the remaining variables are fixed at their current values. This is possible as all variables have a Normal distribution. Algorithm 1 outlines the Gibbs sampler used to generate posterior samples from $P(\eta(\mathbf{x})|\theta, \Lambda(\mathbf{x}))$. Using a Gibbs sampler ensures that the correlation between points remains the same, and computational time is saved by about a third as compared to the full ABC-MCMC.

**Algorithm 1** Gibbs Sampling

Start with initial values $\eta_1^{(0)}, ..., \eta_n^{(0)}$sampled from prior distribution
**for** $i = 1, 2, ...$ **do**
    $\eta_1^{(i)} \sim P(\eta_1^{(i)}|\eta_2^{(i-1)}, \eta_3^{(i-1)}, ..., \eta_n^{(i-1)})$
    $\vdots$
    $\eta_j^{(i)} \sim P(\eta_j^{(i)}|\eta_1^{(i)}, ..., \eta_{j-1}^{(i)}, \eta_{j+1}^{(i-1)}, ..., \eta_n^{(i-1)})$
    $\vdots$
    $\eta_n^{(i)} \sim P(\eta_n^{(i)}|\eta_1^{(i)}, \eta_2^{(i)}, ..., \eta_{n-1}^{(i)})$
**end for**

Before using a Gibbs sampler to generate posterior samples, I attempted to use an alternative method to sample conditionally, in an effort to minimise computation time. This was managed using the Normal conditioning equations (Kotz et al., 2000) shown below:

$$\begin{aligned}
\eta_j|\eta_{j-1}, ..., \eta_1 &\sim \mathcal{N}(\mathbb{E}[\eta_j|\eta_{j-1}, ..., \eta_1], \text{var}[\eta_j|\eta_{j-1}, ..., \eta_1]) \,, \\
\mathbb{E}[\eta_j|\eta_{j-1}, ..., \eta_1] &= \mathbb{E}[\eta_j] + \text{cov}[\eta_j, (\eta_{j-1}, ..., \eta_1)]\text{var}[\eta_{j-1}, ..., \eta_1]^{-1} \begin{bmatrix} \eta_{j-1} - \mathbb{E}[\eta_{j-1}] \\ \vdots \\ \eta_1 - \mathbb{E}[\eta_1] \end{bmatrix}, \\
\text{var}[\eta_j|\eta_{j-1}, ..., \eta_1] &= \text{var}[\eta_j] - \text{cov}[\eta_j, (\eta_{j-1}, ..., \eta_1)]\text{var}[\eta_{j-1}, ..., \eta_1]^{-1} \\
&\quad \times \text{cov}[(\eta_{j-1}, ..., \eta_1), \eta_j] \,.
\end{aligned} \tag{2.3}$$

Equation (2.3) allows us to draw $n$ times from a univariate Gaussian distribution instead of a vector length $n$ from a multivariate distribution. Using this method ensures that the correlation between points remains the same and we minimise computation time. The pseudocode for the method is outlined in algorithm 2, where $s$ is the number of draws taken from the Latent Gaussian process. Instead of rejecting the whole sample, we only reject each individual value for $\eta(\mathbf{x}_i)$ if it does not agree with the corresponding labelling, $\eta(\mathbf{x}_i)$. We never reject any individual $\eta(\mathbf{x}_i)$ values that are shown to agree with their corresponding label. The conditional sample comes from the same target distribution as the joint sample due to all distributions being Normal. The first sampled point is not conditional on anything, but the remainder of the sampled points are conditional on all previous points, and the normal conditioning holds due to Normal theory (Kotz et al., 2000).

I did not persist with the above method because I could not prove that this method was sampling from the same posterior predictive distribution, $P(\eta(\mathbf{x})|\theta, \Lambda(\mathbf{x}))$. To make sure that I was sampling from this distribution, I moved to a Gibbs sampler as everything was Normally distributed and the full conditionals are known.

When using the conditional sampling method outlined above, the ordering of $\mathbf{x} \in D$ becomes very important. If the points are ordered by first including all those with label $l_1$ followed by all those with label, $l_2$, as in Equation (2.3), then the main expense comes from sampling the first point with label $l_2$ (the first time that $\eta$ switches from negative to positive). The computational expense is greatest when the conditional variance from the previous points is very small and the Gaussian process finds it hard to make the jump into positive space. To improve on this, the points are re-ordered as shown in Algorithm 2. There are $n_j$ points known to be in $R_1$ and $(n - n_j)$ points known to be in $R_2$. Distances are calculated and ordered for each pair from $\mathbf{x}_1 \in R_1$ and $\mathbf{x}_2 \in R_2$ (lines 2-3). These pairs, $(\mathbf{x}_1, \mathbf{x}_2)$, are then added into a new matrix, $M$, one at a time starting with the pair having the smallest distance. A pair is not added if one of the points is already included in $M$ (lines 4-9). Once this has been completed for all pairs of points, if there are more points in one of the regions, then these remaining points are added to the end of $M$ (lines 10-14). For

**Algorithm 2** Conditional Sampling

*Re − ordering* :
Calculate Euclidean distance between each $\mathbf{x}_1 \in R_1$ and each $\mathbf{x}_2 \in R_2$
Order pairs, $(\mathbf{x}_1, \mathbf{x}_2)_k \;\; k = 1, ..., n_j(n - n_j),$ according to distance
$M$ is an empty $p \times n$ matrix
**for** $k = 1, ..., n_j(n - n_j)$ **do**
    Select $(\mathbf{x}_1, \mathbf{x}_2)_k$
    **if** $\mathbf{x}_1$ AND $\mathbf{x}_2 \notin M$
        add $\mathbf{x}_1$ AND $\mathbf{x}_2$ to $M$
    **end if**
**end for**
**for** $i = 1, ..., n$ **do**
    **if** $\mathbf{x}_i \notin M$
        add $\mathbf{x}_i$ to $M$
    **end if**
**end for**
*Sampling* :
**for** $i = 1, ..., n$ **do**
    draw $\eta_i^* \sim \mathcal{MVN}\Big[\mathbb{E}[\eta_i|\eta_{i-1}, ..., \eta_1], \text{var}[\eta_i|\eta_{i-1}, ..., \eta_1]\Big]$
    **while** $\text{sign}(\eta_i^*)$ does not agree with $\Lambda(x_i)$ **do**
        draw $\eta_i^* \sim \mathcal{MVN}\Big[\mathbb{E}[\eta_i|\eta_{i-1}, ..., \eta_1], \text{var}[\eta_i|\eta_{i-1}, ..., \eta_1]\Big]$
    **end while**
    $\eta_i = \eta_i^*$
**end for**

the example in the following section we will have 5 points in $R_1$ followed by 7 points in $R_2$. This translates to ordering the points alternately outwards from the interval between $R_1$ and $R_2$. Again, I did not persist with this method since the ordering of the points does not affect the computability of a Gibbs sampler.

The methodology presented in this chapter is designed for classifying and dealing with numerical models that have exactly two solutions. For models where there are more than two solution regions, I would suggest a nested approach by splitting up the problem into a series of binary classifications. For example, say we had a model where we knew there were three separate regions to be classified. I would first apply the method to the regions labelled as Region 1 and not Region 1. Then for the points labelled as not being in Region 1, I would build a separate latent model to further classify the points lying in Regions 2 and 3.

For a future research problem, it would be of interest to examine whether the proposed methodology could be adapted to cope with a larger number of solution

regions. However, given that I have used the negative and positive divide in the latent Gaussian process, this may be challenging.

### 2.2.2 An Illustrative Example in 1 Dimension

To illustrate the concept, I now apply my method to a simple example, the results of which are shown in Figure 2.1. For inputs $x \in [0, 20]$, the true labelling function, $\Lambda$, is set to be:

$$\Lambda(x) = \begin{cases} l_1 & \text{if } x < 7 \\ l_2 & \text{if } x \geq 7. \end{cases}$$

The initial inputs are $\mathbf{x} = \{0, 1, 3, 5, 6, 8, 11, 12, 15, 17, 19, 20\}$, where inputs $\mathbf{x} = \{0, 1, 3, 5, 6\}$ are known to be in $R_1$ and are given label $l_1$, whilst the remaining points, $\mathbf{x} = \{8, 11, 12, 15, 17, 19, 20\}$, are known to be in $R_2$ and so are given label $l_2$. I have chosen a linear prior mean function for this example, and have applied a linear transformation to the inputs to help with the estimation of $\eta$. See Section 2.4 for a discussion on these prior choices.

The latent Gaussian process is estimated following the method outlined in Section 2.2.1. Once a prediction for the labelling function, $\Lambda$, is obtained, the boundary is estimated to be 7.15, as shown by the solid red line in Figure 2.1. This is a reasonable result since the boundary was set to be $x = 7$, and the model was not provided with any knowledge of where the boundary actually lies. If the boundary was actually $x = 6$, then the initial information used to estimate the Gaussian process would not differ. This high level of uncertainty in the results is shown by the credible intervals for the estimate being large, roughly equal to the bounds of $R_1$ and $R_2$ that the example was set up with.

### 2.2.3 Alternative Methodology

In this subsection I will outline two alternative attempts to the problem set out previously. Due to computational constraints, and the associated difficulties of extending to higher dimensions, I do not employ the methods detailed below, but

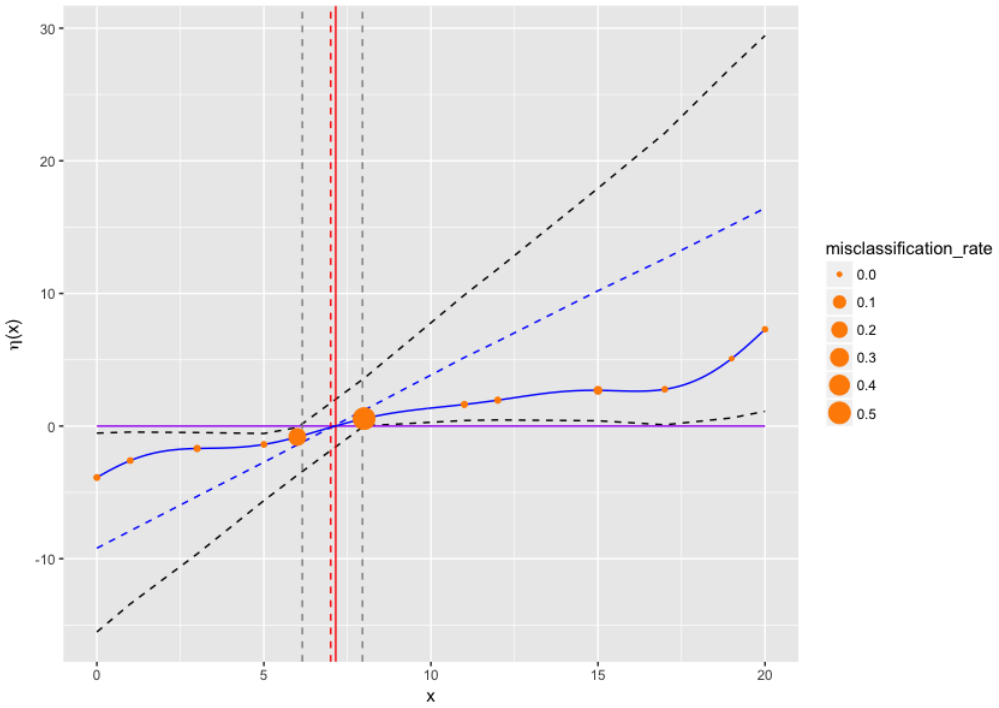



Fig. 2.1 1 dimensional example with 2 output regions. The posterior mean of the latent Gaussian process (solid blue) is shown along with the prior mean (dashed blue), true boundary (dashed red) and boundary estimate (solid red). Both have 95% credible intervals included (black/grey dashed lines). Initial data points are shown in orange with size corresponding to misclassification.

they are included for interested readers. Note that these methods were only developed for a one dimensional example, but could be expanded to higher dimensions.

My initial frequentist approach was to use the EM algorithm (Dempster et al., 1977) to model the latent process as a missing information problem. The EM algorithm is an iterative process that involves two steps: the expectation step (E-step) and the maximisation step (M-step). Carrying out the algorithm will produce not only maximum likelihood estimates of the parameters for a probabilistic model, but also an estimate for the model itself. It was introduced by Dempster et al. (1977) and is a generalisation of maximum likelihood estimation used on incomplete data sets. The general problem with using standard maximum likelihood for incomplete data is due to the log likelihood, $\mathcal{L}(\theta; \mathbf{x})$, having multiple local maxima and no closed form solution (Do and Batzoglou, 2008). By using the EM algorithm, a unique global maxima can be found. The E-step estimates the complete-data by taking the expectation conditional on current maximum likelihood estimates of the model parameters, $\theta$. During the M-step, $\mathcal{L}(\theta; \mathbf{x})$ is maximised to find an estimate for $\theta$

given the current values of the data. The method proceeds by iterating the E and M steps. Convergence to a local maximum is guaranteed (Dempster et al., 1977), but can be slow.

We can apply the EM algorithm to the latent variable problem above to estimate the latent Gaussian process, $\eta(\mathbf{x})$ as well as the parameters, $\theta = (\boldsymbol{\beta}, \sigma, \delta)$. Data are defined as censored if there is only partial information known about its value. It is certain the data lies beyond a boundary point, but it is not known exactly how far above or below (Harrell, 1979; Harrell and Sen, 1979). This is therefore suitable for my problem since we are outlining a latent model where there is no record of its true value. We still keep the labelling function $\Lambda(.)$ as used in Section 2.2.1, but the main difference comes in the modelling of $\Lambda(\mathbf{x})$ when conditioned on the latent Gaussian process. $\Lambda(\mathbf{x})$ can now be modelled using $\eta(\mathbf{x})$, so that:

$$\Lambda(\mathbf{x})|\eta(\mathbf{x}) = \begin{cases} l_1 & \forall\ \mathbf{x} \in D\ :\ \eta(\mathbf{x}) < \psi \\ l_2 & \forall\ \mathbf{x} \in D\ :\ \eta(\mathbf{x}) \geq \psi. \end{cases}$$

Treating the problem as a censored data problem, we no longer use the negative/ positive labelling with a threshold at zero, but instead use the EM algorithm to estimate the threshold boundary between the two regions and label this, $\psi$. The $n_j$ number of points with label $l_1$ (Region 1) are known to take any value below a boundary, $\eta = \psi$, and the remaining $(n - n_j)$ points with label $l_2$ (Region 2) are known to take any value above $\psi$. Each of the inputs are censored differently. The input values never change, it is the value of $\psi$ that changes as the parameters of the Gaussian process change. A threshold can be made here to find the approximated boundary between regions. Hence, it is suitable to fit a censored EM algorithm.

In the likelihood step (M-step), the likelihood is maximised to find the parameters, $\boldsymbol{\beta}$, $\sigma$, $\delta$ and $\psi$. As before, $\boldsymbol{\beta}$ is a vector of unknown coefficients, $\sigma^2$ is the GP variance and $\delta$ is the correlation length parameter. For an uncensored Gaussian process, the likelihood would simply be the multivariate normal distribution, but here it is necessary to incorporate knowledge of the boundary. Specifically, the fact that there are $n_j$ points below $\eta = \psi$, and $(n - n_j)$ points above $\eta = \psi$. Again, we must take

account of the correlation between data points with the likelihood being a single joint Gaussian, rather than the product of $n_j$ normal distributions multiplied by $(n - n_j)$ normal distributions. Therefore, the likelihood becomes:

$$\mathcal{L}(\theta; \mathbf{x}) = \frac{|\mathbf{A}|^{-1/2}(2\pi\sigma^2)^{n/2} \exp\left\{-\frac{1}{2\sigma^2}(\mathbf{x} - \mathbf{H}\boldsymbol{\beta})^T \mathbf{A}^{-1}(\mathbf{x} - \mathbf{H}\boldsymbol{\beta})\right\}}{P(x_1 < \psi)P(x_2 < \psi)\dots P(x_{n_j} < \psi)P(x_{n_j+1} > \psi)\dots P(x_n > \psi)},$$

where $\mathbf{H} = (\mathbf{h}(x_1), \dots, \mathbf{h}(x_n))$, with $\mathbf{h}(x)$ being a vector of basis functions of $x$, and $\mathbf{A}$ is an $n \times n$ covariance matrix with entries $A_{ij} = \sigma^2 c(x_i, x_j)$ as defined in Section 2.2. The numerator is a multivariate Normal density and the denominator takes into account the censoring but is only an approximation due to the data correlation. The denominator treats the points as if they are independent, which is unsuitable for retaining a correlation between input values. The simplification that leads to independence here is considered not to make much difference to the overall approximation of the latent process.

To save computation time, the $\boldsymbol{\beta}$ parameters in the likelihood are set to zero (constant), adding another degree of freedom. One of the largest contributors to computation time is the maximisation of the length parameter, $\delta$. To overcome this problem, I do not incorporate it in the same maximisation as $\sigma$ and $\psi$, and is instead only calculated every 4 or 5 iterations. This greatly reduces computation time with little change in the final latent process approximation. $\delta$ is also transformed using a logarithmic transformation, $\tau = 2\log(\delta)$ before the likelihood is maximised using maximum likelihood. Another common tool used to speed up maximisation is to differentiate the likelihood analytically and use the gradient to aide the maximisation. Due to the complexity of the likelihood, this is not attempted here.

The expectation step (E-step) is much simpler than the maximisation step. We need to calculate the expected value of the points from the latent Gaussian process, $\eta$, given the current estimate of the parameters, $\theta$, and known input values, $\mathbf{x}$. For the EM algorithm, it is necessary to give initial values for the $\eta(\mathbf{x})$ estimates. These are entirely arbitrary, but I have chosen to let $\eta(\mathbf{x}) = -1$ for those labelled $l_1$ and $\eta(\mathbf{x}) = 1$ for those with label $l_2$. Once actual values for the latent process have been estimated from the expectation step, the maximisation of the likelihood should be

improved. This is due to Gaussian processes not coping well with modelling step functions, which are obtained here with the initial $\eta(\mathbf{x}) = -1/1$ values.

The process iterates between the E-step and the M-step until the difference between consecutive parameter estimates is smaller than a specified tolerance. The EM algorithm produces the estimated values for the latent Gaussian process at the end of the iterations. This is as well as estimating the parameters for the GP and the boundary, where the thresholding is done automatically.

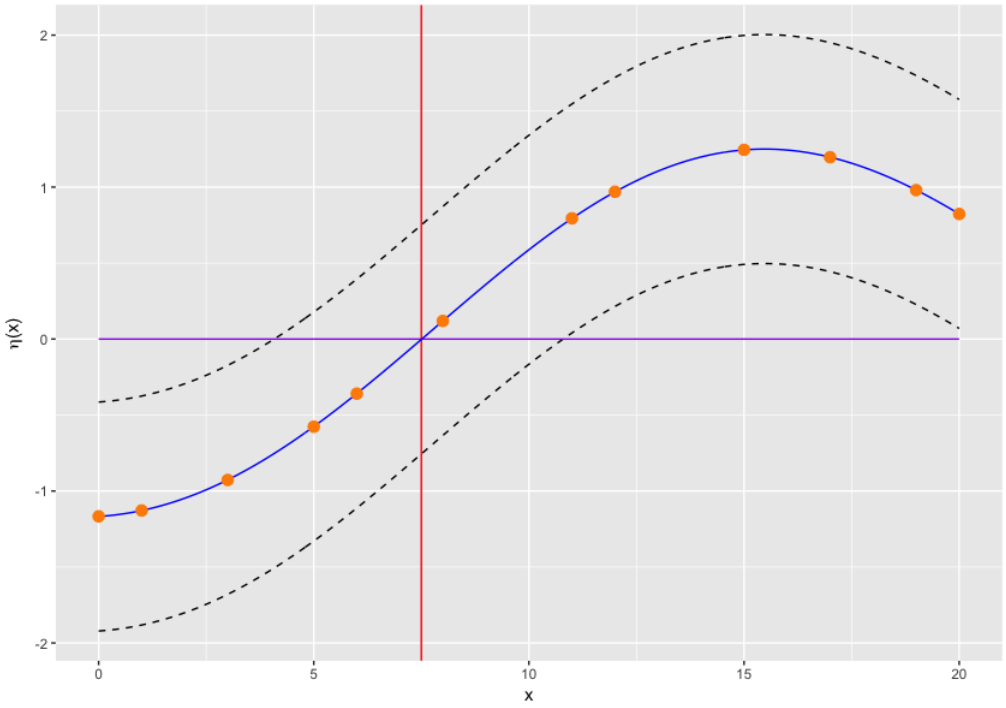



Fig. 2.2 Example from Figure 2.1 using EM algorithm. Posterior mean for the latent Gaussian process (blue) along with uncertainty bounds (dashed) and boundary estimate (red).

The simple example from Section 2.2.2 is now used to illustrate how the censored EM algorithm is applied to the problem. The inputs are, again, a vector of 12 integers ranging between 0 and 20, with the undefined boundary situated in the region $[6, 8]$. As before, those points in the range $[0, 6]$ are given class label $l_1$ (region 1) and the points in the range $[8, 20]$ are given class label $l_2$ (region 2). By making the assumption $\boldsymbol{\beta} = 0$, the algorithm produces an estimate for the parameters $\psi, \sigma^2, \delta$, as well as the latent GP, $\eta(\mathbf{x})$. With starting values $\psi = 0, \sigma^2 = 1, \delta = 1$, the process iterates between the E-step and M-step until the difference between consecutive $\psi$ values is smaller than $10^{-5}$. The latent Gaussian process is shown in Figure 2.2 along

with the estimate for the region boundary. The GP is thresholded at $\psi = -0.123$, resulting in the boundary lying at $x = 7.50$.

Overall, the EM algorithm seems to find the boundary sufficiently well, only being slightly higher than the estimate in Figure 2.1 at $x = 7.15$. However, the method does estimate the latent Gaussian process quite badly as shown in the uncertainty bounds in figure 2.2. I would favour the Metropolis Hastings method over the EM algorithm due to slow convergence and difficulties in expanding to higher dimensions.

I also considered other Bayesian methods, for which an alternative to Metropolis Hastings (used in Section 2.2.1) is approximate Bayesian computation (ABC) (Turner and Van Zandt, 2012; Wilkinson, 2008). This is specifically useful when the likelihood cannot be fully specified and so standard methods of Bayesian estimation are difficult to apply. ABC works by simulating predicted model data and comparing with the known observations to estimate the posterior distribution for the model parameters. The most basic form of ABC is based on a rejection algorithm where a set of parameters, $\theta$, are drawn from a prior distribution $\pi(\theta)$. These parameters are then used to simulate data $X$ from the unknown model, $\eta(\theta)$, and are accepted or rejected according to a given comparison criteria with the observations, $Y$. Here, $\theta$ is accepted if $\rho(X, Y) \leq \delta$, where $\rho(.,.)$ is a measure of distance, usually taken to be the Euclidean distance, $\|X - Y\|$ (Turner and Van Zandt, 2012). All accepted values of $\theta$ come from an approximation to the true posterior distribution. $\delta$ is a small quantity that specifies how close the approximation is to the true posterior distribution. When $\delta = 0$, the distributions are equal, but as $\delta \to \infty$ the approximation then becomes equal to the prior distribution (Wilkinson, 2008).

I use a version of ABC-MCMC to estimate $\eta(.)$, given the labelling function, $\Lambda$, that I defined for the Metropolis Hastings version in equation (2.1). I no longer estimate the threshold boundary as in the EM version, but return to the specifications made in equation (2.2) where we model $\Lambda(.)$ using $\eta(.)$ thresholded at zero. Hence, by applying this method, we obtain an estimate for the latent GP parameters, $\theta = (\boldsymbol{\beta}, \sigma, \delta)$ that generates draws that are negative in all input space of the region labelled $l_1$ and positive in the input space of the region labelled $l_2$.

The algorithm starts with an initial value $\theta_0$ for $\theta$, then a candidate, $\theta^*$, is proposed from a prior distribution $\pi(\theta)$. $\theta^*$ is used to simulate data from the GP, which is then accepted or rejected according to specified criteria. To ensure the Gaussian process meets the criteria, $\theta^*$ is accepted if the signs of the estimated data and observations agree, and then $\theta_1 = \theta^*$. If not, $\theta^*$ is rejected and $\theta_1 = \theta_0$. This process continues iteratively until there is a chain of values $\{\theta_0, \theta_1, \ldots, \theta_m\}$ that form a sample from the posterior distribution, where $m$ is the number of iterations of the algorithm. Hence on the $(i+1)^{th}$ iteration, $\theta^*$ is simulated from $\pi(\theta)$. If accepted we set $\theta_{i+1} = \theta^*$, otherwise $\theta_{i+1} = \theta_i$.

To illustrate the concept, the same example from the Metropolis Hastings (Figure 2.1) and EM algorithm (Figure 2.2) is shown in Figure 2.3. As in Section 2.2.2, the points in Region 1 are given the label $l_1$ and the points in Region 2 are given the label $l_2$ and the latent GP is found conditional on this. At each draw from the prior distribution, a sample of $\theta = (\beta, \sigma^2, \delta)$ is taken and accepted or rejected according to the given criteria. If the sampled latent GP, $\eta$, is negative for values in Region 1, and positive for values in Region 2, the proposed $\theta$ is accepted. Alternatively $\theta$ is rejected and a new proposal is drawn from the prior. After an estimate for the latent process has been found, it is then thresholded at $\eta = 0$ to give a value of $x$ for where the boundary between regions lies. I find this to be $x = 7.05$, which is shown by the red line in Figure 2.3. This is a reasonable estimate that is similar to the estimates found from using the Metropolis Hastings and EM algorithms.

Overall, I have found that the Metropolis Hastings methodology is best suited for my problem since it involves the likelihood, making the estimate far more accurate. As well as this, it avoids the large rejection rate involved in using ABC and can be easily expanded to higher dimensions. As the dimension increases, the rejection rate in ABC also increases rapidly, making Metropolis Hastings a far more efficient method.

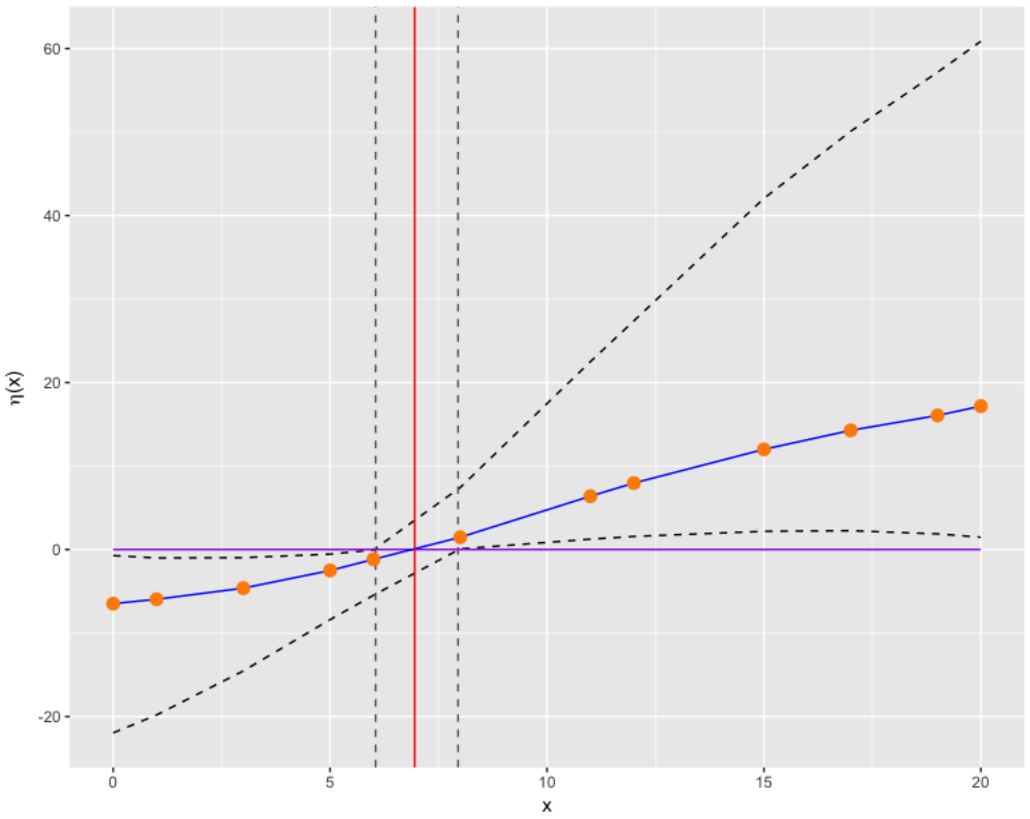



Fig. 2.3 Example from Figure 2.1 using ABC algorithm. Posterior mean for the latent Gaussian process (blue) shown along with the boundary estimate (red) and 95% credible intervals for both (dashed). Initial data points are shown in orange.

## 2.3 Misclassification

The method of model validation I use is based on a leave-one-out cross-validation, where the pseudocode is given in Algorithm 3. In uncertainty quantification, this usually involves leaving each training point out in turn, fitting a Gaussian process to the remaining points, and then using this to predict the point that was left out (Rasmussen and Williams, 2006). Given that my method models the class labelling function, $\Lambda$, I adapt this slightly to calculate a misclassification rate to see which of the initial inputs are more likely to be influential to the classification prediction. A leave-one-out cross-validation is performed on samples taken from $P(\eta(\mathbf{x})|\beta, \sigma^2, \delta, \Lambda(\mathbf{x}))$ to predict the class label of each point left out in turn. From these samples, I calculate the proportion of times each point is misclassified. Points that have a large misclassification rate are likely to give an indication that the surrounding areas will have high uncertainty when making classifications. We expect points close to the boundary between $R_1$ and $R_2$ to have a high misclassification rate,

and points where there are many neighbouring points (where we have high levels of information) to have a low misclassification rate.

**Algorithm 3** Misclassification Rate Using Leave-one-out Cross-Validation

```
Leave-one-out cross validation:
C is a zero vector of length n
for j = 1, ..., 10000 do
    for i = 1, ..., n do
        Let x^(j) be a sample from P(η(x)|β, σ², δ, Λ(x))
        x* = x_1^(j), ..., x_{i-1}^(j), x_{i+1}^(j), ..., x_n^(j)
        Fit η(x*)|Λ(x*) ~ GP(m(x*), v(x*, x'*))
        Predict Λ(x_i^(j))|η(x*)
        if Λ(x_i^(j)) ≠ Λ(x_i) do
            C_i = C_i + 1
        end if
    end for
end for
Misclassification rate = C/10000
```

Leave-one-out validation applied to the example is Section 2.2.2 in shown in Figure 2.1, where the size of the data points corresponds to the rate of misclassification. As expected, the rate is largest for the two points either side of the boundary. In a 1-d example such as this, these points are the most critical since they are the points that restrict the boundary to a precise region of input space. It is also interesting to note here that the remaining points have a misclassification rate of almost (but not quite) zero. In fact, we can see that this is due to the latent process occasionally crossing the axis. For each point in this example, I find that roughly this happens for every 1 in 250 samples and an example draw is shown in Figure 2.4. This is caused by both a large gap between points and the Gaussian process having a short correlation length parameter, leading to the latent process having the chance to bend quickly over the $\eta = 0$ threshold between known points in the same region. I have not specified that the latent model only has one boundary. This highlights the fact that prior choices can have a large effect on a method such as this with minimal initial information. I discuss this further in Section 2.4.

Figure 2.5 shows the result of fitting a Gaussian process to the misclassification rate from Figure 2.1. This means that we can predict the misclassification rate for any point and get a general idea for the overall misclassification across the input

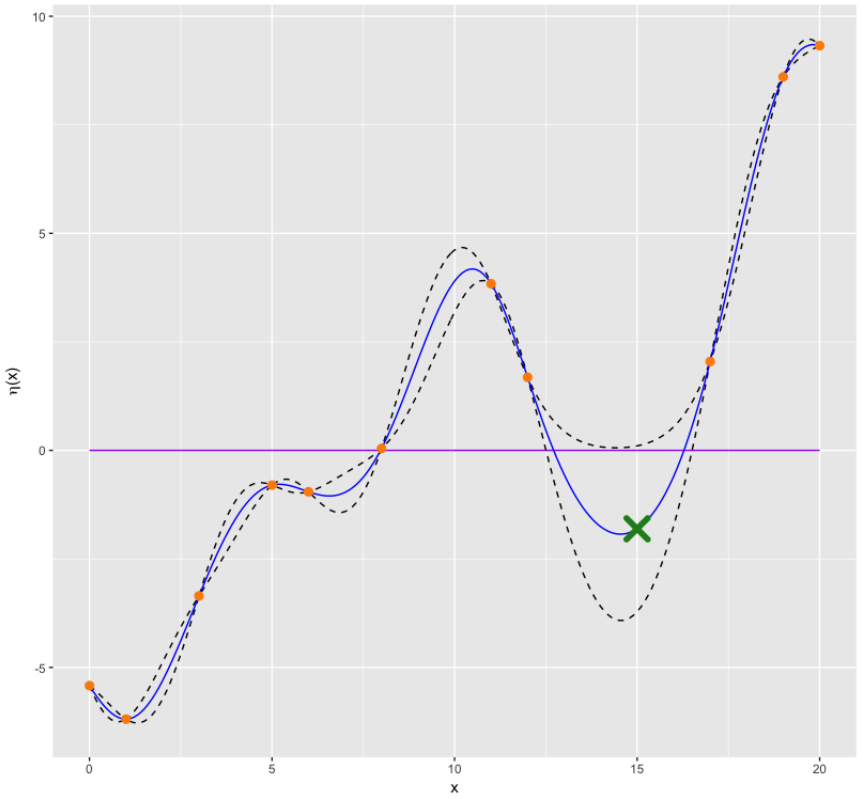



Fig. 2.4 Plot to show an example of a draw from the latent Gaussian process in Figure 2.1 when one point is left out in a leave-one out cross validation. The point, $x = 15$, is left out and a GP (blue) is fitted to the remaining points shown in orange. This is then used to predict the point left out (green). This plot shows a rare case when the point left out is given the wrong sign and classification.

space. When performing this, I had to make sure to transform the data so that I did not get any negative misclassification for any points. I could not use a log transformation because this could produce zero misclassification in some places; therefore I transformed the data by taking the square roots. The plot is as expected; a large misclassification across the boundary where we have no knowledge of the classification of points, then almost zero for the rest of the input space where we have more information.

## 2.4 Prior Choice Methodology

Based on the example in Section 2.2.2, it is clear that it is important to place suitable priors on the model parameters and the prior mean function (Currin et al., 1991). One interesting aspect of Gaussian processes is their behaviour in the far edges of the input space. As Gaussian processes get far away from any data, they revert to

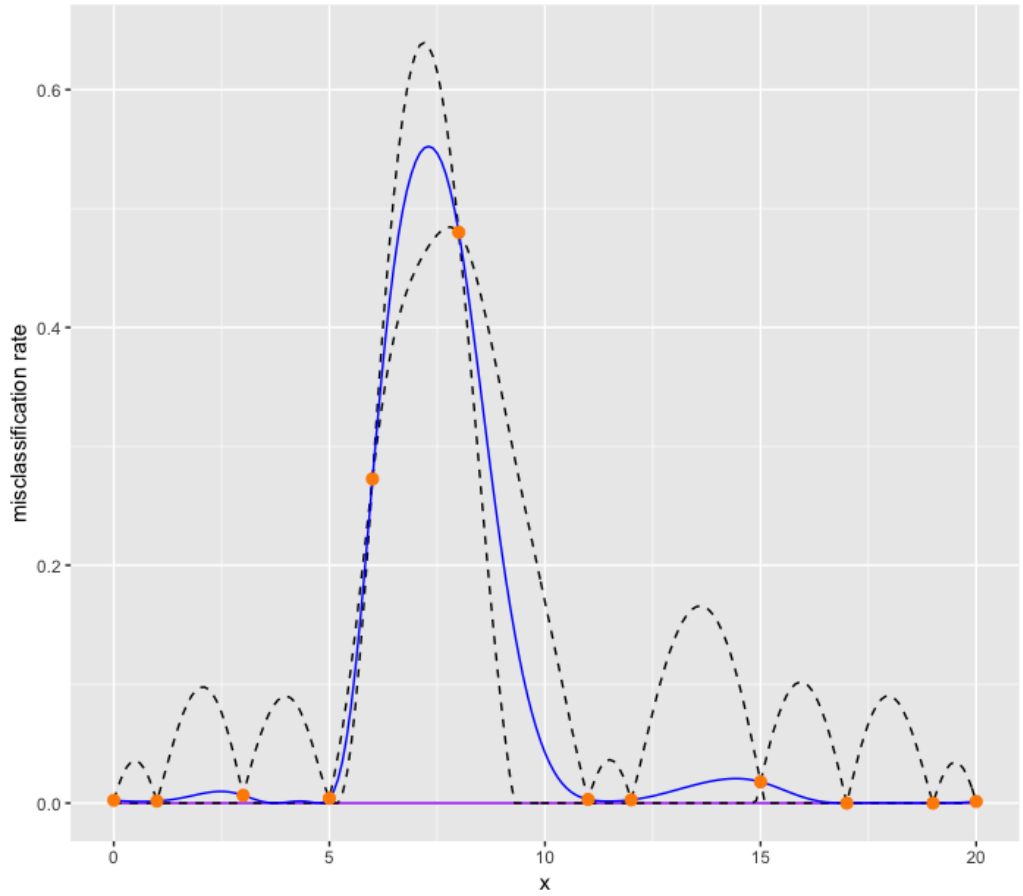



Fig. 2.5 Gaussian process fitted to the misclassification rate in Figure 2.1. The expected mean is shown in blue, with original training point (orange) and uncertainty (dashed). A square root transformation is applied to ensure a negative misclassification does not occur.

the prior mean, $m$ (Sacks et al., 1989). This would be a problem for, say, a constant prior mean. If a constant mean function is placed on the Gaussian process, then we start to observe the overall latent process tending towards the horizontal prior mean in the edges of our input space. Given that I have chosen $\eta$ to be negative or positive according to the labelling, a constant prior mean may be estimated to be close to $m = 0$ and we find that it is very easy for the process to switch signs, forcing a misclassification.

In many cases, we will have extra information in the initial data that will help us to choose a more appropriate prior mean function. For example, we might use a prior mean function based on whether both regions are simply connected, or on the number of times the latent process is expected to change signs over the whole input space (for example by using a polynomial of that degree). Hence, any expert knowledge from the system modeller is very useful, particularly if they know how

many distinct regions are expected. In a situation such as that in Figure 2.1, I had extra knowledge that there were only two output regions and so it should be the case for the latent variable to only cross the $x$-axis once in the input space. I thus used a linear mean function which forced the latent Gaussian process away from the $x$-axis and bound $\eta(x)$ away from zero near the edges of the domain.

The right plot in Figure 2.6 shows the effects of choosing a constant mean function for the 1-d example. Although the boundary estimate between $R_1$ and $R_2$ is almost the same as in the linear version in Figure 2.1, the latent process is significantly different. We can clearly see that the Gaussian process is returning to the prior mean (dashed blue) at the edges of the plot. Further, the misclassification rates are also much larger here, confirming that we are much more likely to see misclassifications when using a constant mean function.

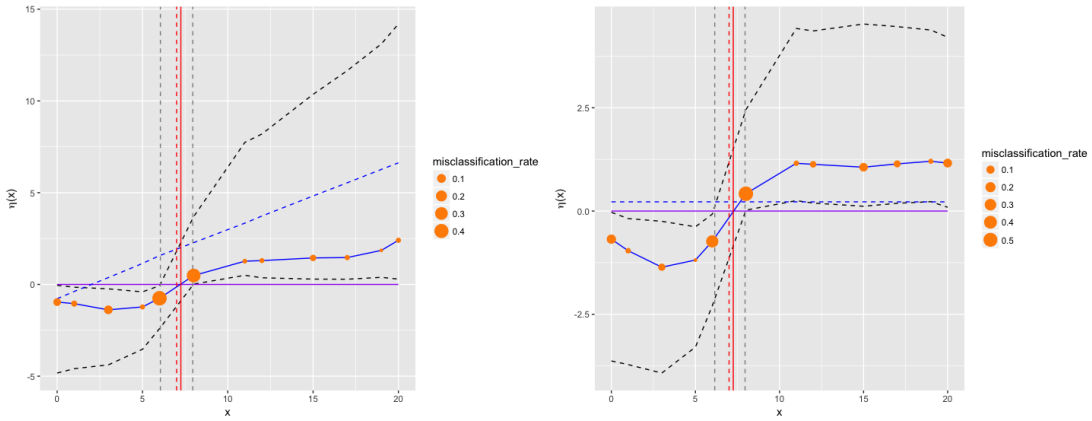



Fig. 2.6 Same example as of Figure 2.1 but with some prior changes. The left plot is where the data are not transformed and the prior mean (blue dashed) crosses close to the origin (0,0). The right plot shows the effect of choosing a constant prior mean function.

If instead, one region was split either side of the other region as shown in Figure 2.7, then it would be sensible to place a quadratic prior on the mean function, ensuring the Gaussian process would not return to the $x$-axis (where $\eta(x) = 0$) in the extreme values of the input space of the outer region. The plot in Figure 2.7 is a simple 1-d example where the left and right sections are region 1 ($\Lambda(x) = l_1$) and the middle section is region 2 ($\Lambda(x) = l_2$). For inputs $x \in [0, 20]$, I set the true labelling

function, $\Lambda$, to be the following:

$$\Lambda(x) = \begin{cases} l_1 & \text{if } x < 4 \text{ OR } x > 14 \\ l_2 & \text{if } 4 \geq x \leq 14. \end{cases}$$

The orange points show the initial input points for both regions with their size denoting the corresponding misclassification rate. For the right plot in Figure 2.7, the latent Gaussian process was estimated using a quadratic prior mean function, whilst the left plot was fitted using a constant mean. If we compare the two, we can clearly see that the latent GP in the constant version is beginning to curl up at the edges as it attempts to return to the expected mean which is close to the axis. This does not happen when we use a quadratic mean (right figure) as we have the extra constraint in the prior. Thus, we are much more likely to find a misclassification in the constant version than the quadratic (this is confirmed by the size of the misclassification rates). The constant plot has much larger misclassification rates in the two edge points due to the GP attempting to return to the mean, whilst the two edge points for the quadratic plot have a zero misclassification rate. Although the points either side of the boundaries for both the plots are by far the most misclassified (being the largest), they are overall smaller in the right plot. This is likely to be due to the quadratic shape I am enforcing, as it must cross the axis twice over the input range.

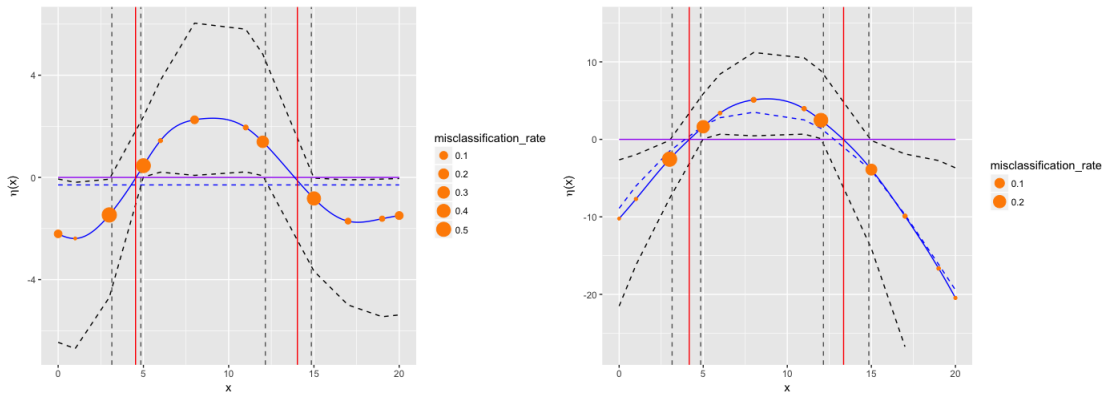



Fig. 2.7 1-d example where region 1 is split either side of region 2. The initial points are shown (orange) along with the expected latent Gaussian process (blue), estimated borders between regions (red) and uncertainty for both (dashed). Misclassification rates are shown by the size of the initial data points. The left plot shows the example with a constant prior mean and the right plot shows the example with a quadratic prior mean function. These are shown by the blue dashed lines in both plots.

In two dimensions, if we had a circle as one region, and the outer remaining space as the other region, then we could choose to fit a quadratic mean function. Or, if we had two distinct circles as one of the regions and the outer remaining space was the other region, then we could consider fitting a quartic mean function.

Although these would appear to be sensible choices, polynomials of a higher order come with a larger number of estimated parameters. Therefore, we should consider whether the classification in the edges of our input space away from the data is useful or not. It may be far more computationally expensive to calculate a high number of parameters and introduce more uncertainty than to assess whether the classification is accurate or not at the edges. As a rule of thumb, I would suggest choosing either a linear or quadratic mean function if these are sensible choices. However, if it is decided that a higher polynomial would better suit the problem, but estimating the latent Gaussian process becomes far too complicated then I suggest using a constant mean. It is important to be careful about any potential misclassifications. If it is such that the number of data points is large, then I leave this up to the reader's judgement on whether fitting a higher order polynomial is worthwhile or not. One could fit Voronoi polygons to the original data to get a rough idea of the number of regions and the general shape, especially if there is little known information from the expert.

If the choice is made to use either a linear or quadratic prior mean function, then there is more prior information that we can use to help with the computation. The estimated boundary between $R_1$ and $R_2$ becomes equivalent to the corresponding $\mathbf{x} \in D$ such that it is the solution to $\eta(\mathbf{x}) = 0$. Therefore, the latent process, $\eta(\mathbf{x})$, must cross the $x$-axis at approximately the boundary between $R_1$ and $R_2$ and we can incorporate this into the prior knowledge of our model. Specifically, this will help approximate the parameter that controls where the latent process crosses the vertical axis more efficiently.

As described in Section 2.2.2, a transformation was applied to the input points so that the boundary between regions in the $x$-axis was approximately at zero in the vertical axis. With this transformation, a tight prior could be placed over the

axis intercept parameter, ensuring the latent process crosses the axis at zero. If we contrast the plot in Figure 2.6 (left) compared with that in Figure 2.1, we notice a significant difference in the resulting latent process. The prior means for each plot are shown with dashed blue lines. Figure 2.1 uses the transformed data and is shown to have an expected mean Gaussian process which follows its prior direction. Figure 2.6 (left) does not include the transformation and is shown to differ by the posterior estimate in region 1 levelling out as it approaches zero. This is clearly not appropriate since (in this simple two region example with no information of the system input) we would expect both sides of the latent process to match. This shows that the transformation in the data can greatly improve the estimate in the latent process and any predictions that would follow.

With the consideration of prior knowledge, it is also important to choose a good estimate for the correlation length parameter, $\delta$ (Oakley and O'Hagan, 2002). The correlation length parameter determines how much the Gaussian process is allowed to bend between each of the initial data points (Rasmussen and Williams, 2006). In particular, if we consider the 1-d example in Figure 2.1, we know that there is only one boundary where the latent Gaussian process is not expected to change sign between data points (apart from at the boundary between regions). If the correlation lengths are allowed to become too small, then there is a chance that the Gaussian process would be able to curve round quickly and briefly incorrectly change sign, causing a misclassification of regions in some input areas. To ensure this does not happen, inverse gamma priors are placed on the $\delta$'s so that they are forced away from zero and being too small. The mean of the prior distribution is kept away from being zero, whilst the scale is kept large to increase the spread. An inverse gamma prior is also placed on the variance, $\sigma^2$. This is a common choice in literature which only allows positive values that are not too close to zero, adding extra flexibility to the model.

## 2.5 Examples in Higher Dimensions

I will now expand my method to two dimensions, and look at an example similar in structure to that seen in Example 2.1. The output is shown in Figure 2.9, where 20 input points, $(x_1, x_2)$, are generated using a Latin hypercube (Welch et al., 1992) over the region $[-1, 7]^2$. I have specified the boundary between $R_1$ and $R_2$ in this example to be the line $x_1 = 3$ (shown in red) so that the true labelling function becomes:

$$\Lambda(x_1, x_2) = \begin{cases} l_1 & \text{if } x_1 < 3 \\ \\ l_2 & \text{if } x_1 \geq 3. \end{cases}$$

In the figure, the yellow points are those initial data points in $R_1$ with label $l_1$ (input space $x_1 < 3$) and the purple points are those initial points in $R_2$ with label $l_2$ (input space $x_1 \geq 3$). The latent Gaussian process has been applied to a grid of points over the input space to show the estimated classification labellings. Two possible draws from the latent Gaussian process are shown in Figure 2.8.

To show uncertainty within the 2-d example, Figure 2.9 shows the probability of input points being classified into $R_1$ compared with $R_2$. The dark blue regions represent high probability of being classified into $R_1$ and the light blue represents high probability of being classified into $R_2$. A misclassification rate is calculated for each point as described in Section 2.3 and is shown in Figure 2.9. As expected, the points near the boundary have a larger rate of misclassification and the uncertainty increases.

I chose to fit the Gaussian process with a linear prior mean because I know that the boundary is parallel to the $x_2$-axis. Even if we didn't know this apriori, we can see from the initial data that a linear mean might be a reasonable choice since there only seems to be one change in region over the space filling design.

The method outlined in Section 2.2.1 can be further extended to three (or higher) dimensions. As an example of this, Figure 2.10 shows two examples of my method being applied to a 3 dimensional problem. I will not go into great detail about these plots (since it is difficult to plot examples in 3-d), but they are included to demonstrate the model's capabilities. The left plot is a 3-d extension to the plot

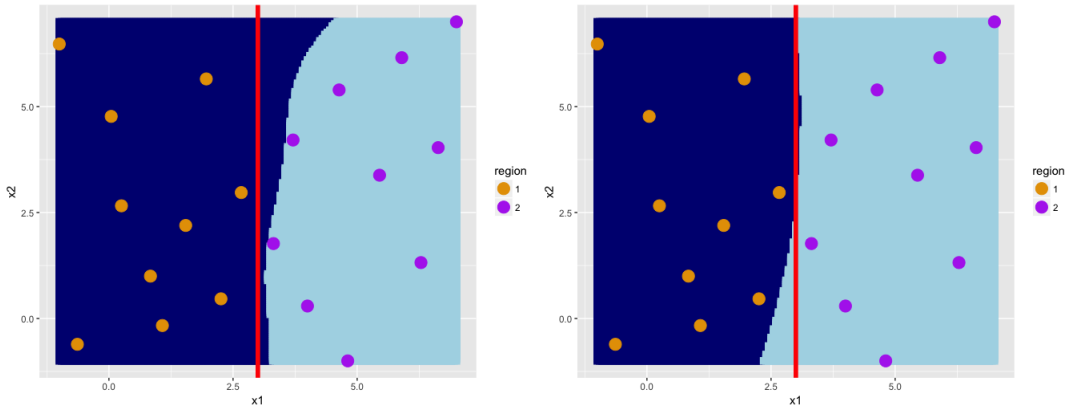



Fig. 2.8 Two different draws of $\Lambda$ for the 2 dimensional example where the two region are split by an $x_1 = 3$ plane (red). The dark blue region corresponds to being classified into $R_1$, whilst light blue corresponds to being classified into $R_2$.

shown in Figure 2.9 where the cuboid is split into two regions by the plane given by $x_1 = 3$. The right plot is slightly more complicated where one region is a sphere and the other region is the outer remaining input space. My method has been successful in finding the boundary in both of these cases.

## 2.6 Comparison with Existing Methods

### 2.6.1 Logistic Regression

Using the 2-d example described in Section 2.5, I will now illustrate the strengths of my method by comparing it with existing methods. One of the most widely used methods for this type of classification is logistic regression (Diggle et al., 1998; Hilbe, 2009; Kleinbaum and Klein, 1994). Similar methods in uncertainty quantification literature are given in Chang et al. (2016) and Salter et al. (2019).

The outputs of using logistic regression for the 2-d example are shown in Figure 2.11, where the logistic model is stated to be the following:

$$\Lambda(\mathbf{x}) \sim \text{Bernoulli}(\eta(\mathbf{x})), \qquad \text{logit}\left(\eta(\mathbf{x})\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2.$$

The bottom right plot shows the underlying probability of being classified into $R_1$. This is a smooth function that shows high probability of being sorted into either of the regions where expected. Two samples are shown in the top two plots in Figure

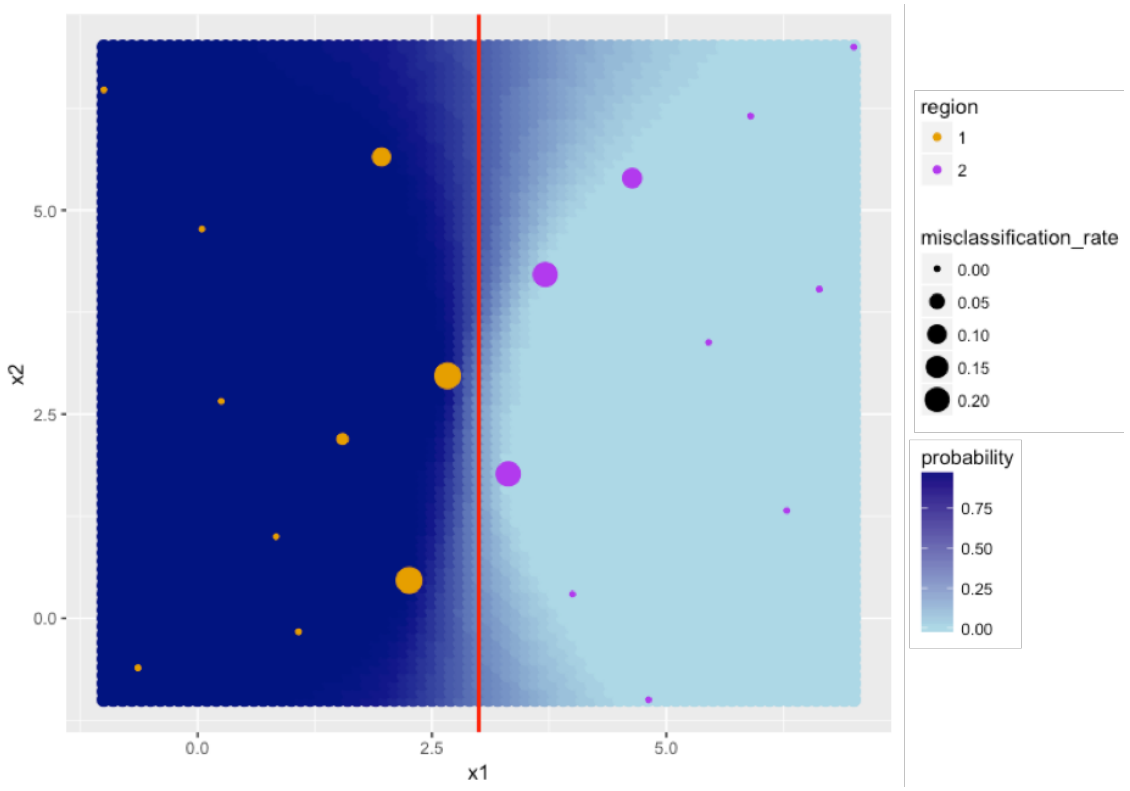



Fig. 2.9 2 dimensional example where the two region are split by an $x_1 = 3$ plane (red). The dark blue region corresponds to a high probability of be classified into $R_1$, whilst light blue corresponds to high probability of being classified into $R_2$. A misclassification rate is also shown based on point size.

2.11, computed using the geoRglm package in R. This exposes the main flaws of using logistic regression for this framework since we can clearly see that because the random Bernoulli sample does not take into account the distance correlation between points, the sampled $\Lambda(\mathbf{x})$ field is not smooth. In practice, $E[\Lambda(\mathbf{x})]$ might be used as a classifier, but with my method, all samples give a coherent full classification. There is no distinct boundary between $R_1$ and $R_2$. Comparing to samples drawn using my method in Figure 2.8, I am able to produce a clean cut boundary in every sample.

The main problem when using logistic regression is the lack of correlation when sampling from the Bernoulli distribution, whereas my method is able to classify jointly over the input space. The underlying probability function retains the correlation structure, but when we sample marginally over all points, all of this is lost. We could run a smoother over these samples, but this can get very complicated and we still would not be able to define the exact boundary between regions. Alternatively, we could take a threshold on the probability function, but it is not clear what value we would choose for this threshold. To make a fairer comparison to my method, the

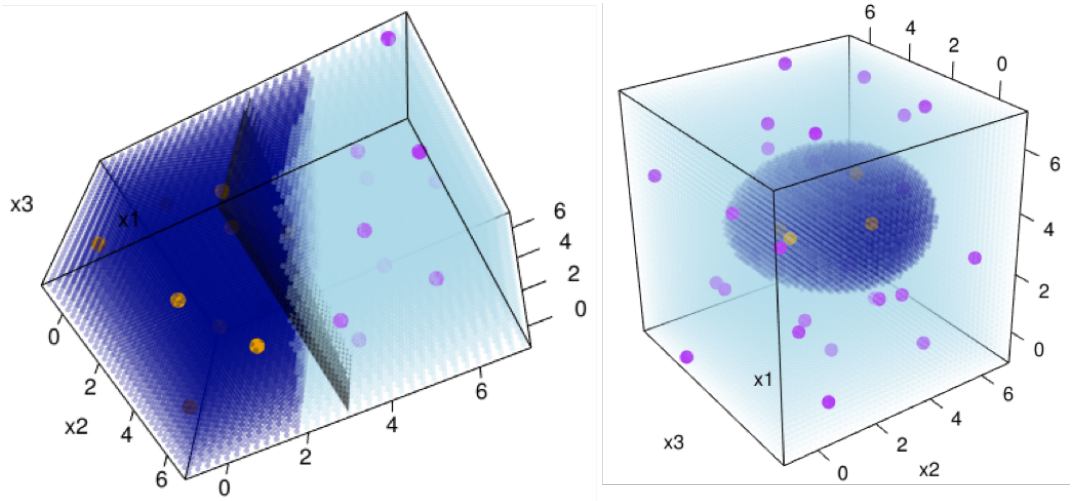



Fig. 2.10 Two examples of applying the method from Section 2.2.1 to 3 dimensional problems. The orange and purple points show the initial points from regions 1 and 2 and the dark blue and light blue show the classification estimates for the input space for regions 1 and 2 respectively.

bottom left plot in Figure 2.11, averages over 1000 Bernoulli samples. This now has the smoothness of the probability function, and is similar to Figure 2.6, but still does not provide an estimate of the boundary. Since I am able to define a boundary estimate, I am therefore able to classify a set of points at the same time rather than only individual points. It is also important to note that we can generate samples that have a clean cut boundary. This can be important when trying to visualise the possible extreme boundary partitions that can be valid from the initial data.

### 2.6.2 Voronoi Tessellation

Figure 2.12, shows a naive approach to this problem by splitting up $R_1$ and $R_2$ using Voronoi tessellation (Gallier, 2008; Kim et al., 2005). We can clearly see that this outperforms the classifications made using logistic regression, but again has several flaws. One of the main problems is that it is only able to classify along arbitrary smooth curves, which means we lose a certain amount of precision compared to my method. Another point to make is that with Voronoi tessellation, we just take the mid-point between the closest points in $R_1$ and the closest points in $R_2$. My method is able to learn more from all initial points. This is shown in Figure 2.6 where the upper section of the boundary estimate is shown to curve far more into $R_2$ than that

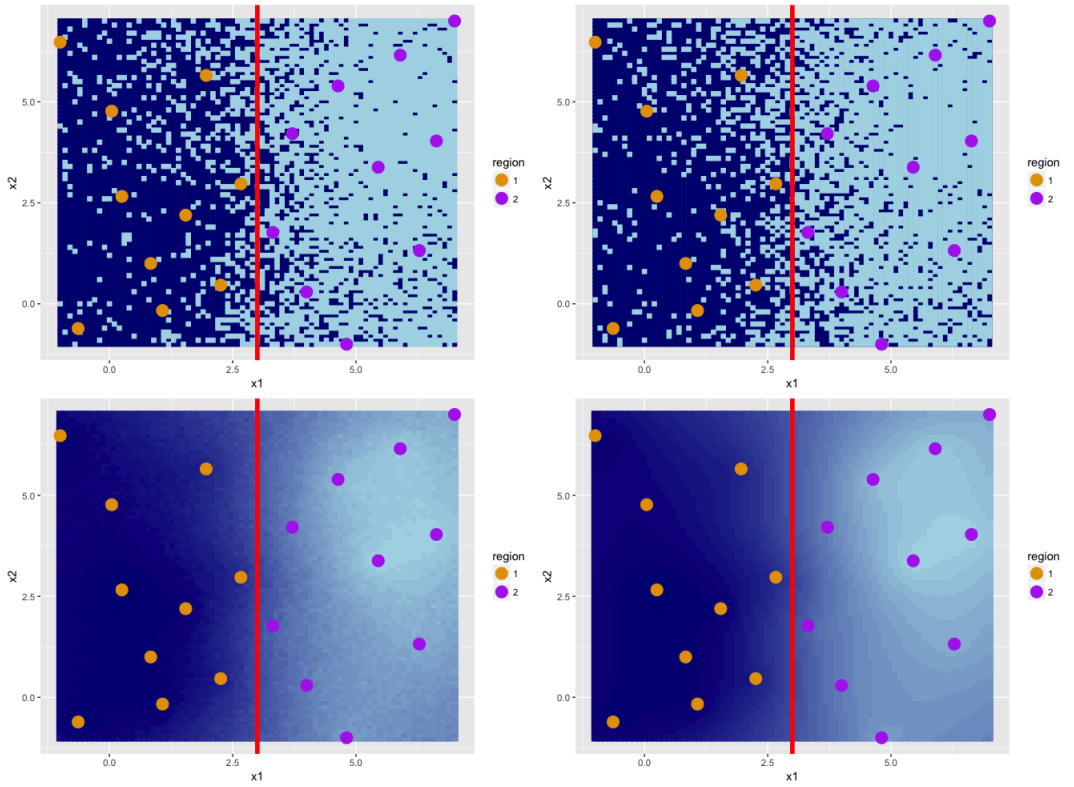



Fig. 2.11 2 dimensional example from Section 2.5 modelled using logistic regression. Top row: Bernoulli samples of region classifcations using logistic regression. Bottom left: average of 1000 Bernoulli samples. Bottom right: underlying probability function of being classified into $R_1$ or $R_2$.

of the lower section. Similarly, we can not make any uncertainty statements with this method.

### 2.6.3 Classification Using Contours

Ranjan et al. (2008) propose an alternative method to logistic regression by attempting to model the boundary between the two separate output regions, specifically as a contour. They use an improvement function in estimating a contour, $S(a) = \{x : y(x) = a\}$, where $a$ is the value of the response surface. A relatively small designed experiment is performed and points are chosen sequentially based on the improvement function weighted towards choosing points on or near the estimate of the contour, (i.e. choose new points, $x$, where $\hat{y}(x)$ belongs to a neighbourhood, $(a - \epsilon, a + \epsilon)$, of the current contour estimate), or where the predicted variance is high. The improvement

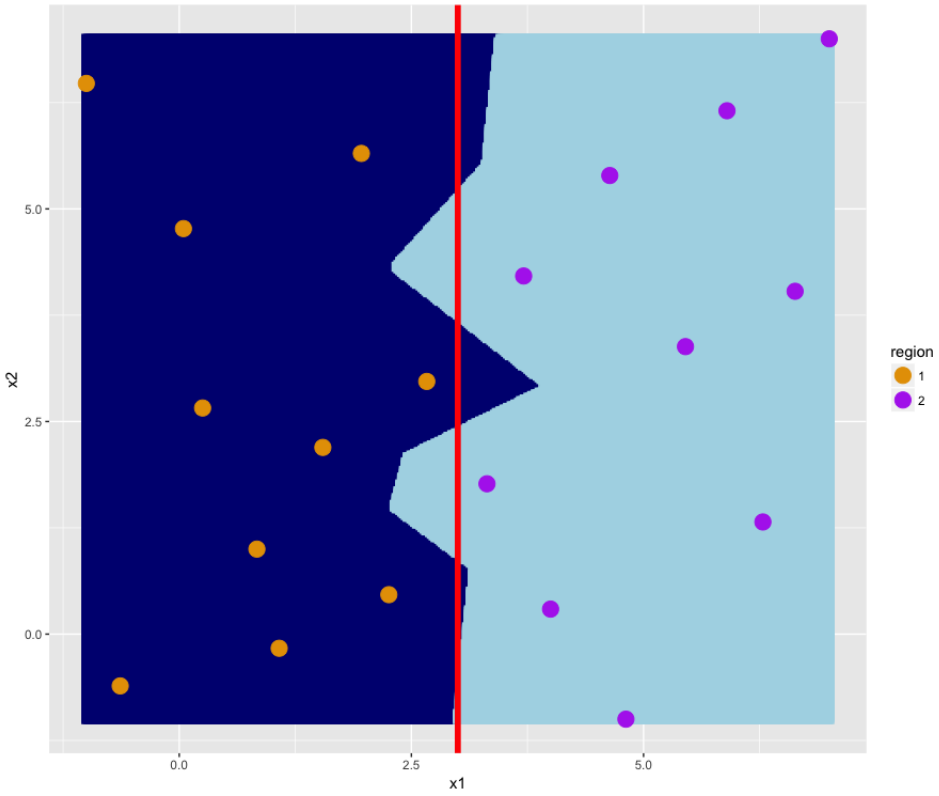



Fig. 2.12 2 dimensional example from Section 2.5 where classifications have been made using Voronoi tessellations.

function has the following form:

$$I(x) = \epsilon^2(x) - \min\{(y(x) - a)^2, \epsilon^2(x)\}, \tag{2.4}$$

where $\epsilon(x) = \alpha s(x)$ for some positive constant $\alpha$ and $y(x) \sim \mathcal{N}(\hat{y}(x), s^2(x))$. The term, $\epsilon$, defines a neighbourhood around the contour that is a function of the standard deviation $s(x)$. This process is aided with the use of Gaussian process emulation. To begin the process of estimating a contour, a small design (25-35%) is first obtained to outline the response surface. The GP is then calculated, and the model parameter estimates are used to evaluate $E[I(x)]$ on the sample space. A new run of the computer simulation is performed at the optimum location of $E[I(x)]$, and the GP is updated. This process is repeated until the budget of allocated simulator runs is exhausted and the final contour estimate, $S(a)$, is extracted.

Although this method appears to be ideal in calculating uncertainty, it requires an underlying smoothness assumption. The whole output space is modelled by one single Gaussian process, where there is a simplifying assumption of the response

surface being smooth in the form of the covariance function. We cannot guarantee that there will not be a discontinuity between the regions, meaning that this method may be unsuitable in some cases. Further, it is likely to become increasingly complex in higher dimensions.

### 2.6.4 Classification Using History Matching

A process known as history matching is used in a method developed by Caiado and Goldstein (2015). History matching is an iterative process designed to reduce the input space of the simulator such that input values that are not likely to result in the observed data are discarded (Andrianakis et al., 2015; Craig et al., 1996; Vernon et al., 2010). Here, it is used to sort data into the separate output regions by discarding regions which are unlikely based on an implausibility criterion. The main feature of the process is the use of an implausibility measure, $I$, which uses a metric based on the number of standard deviations between the observed data, $z_i$, and the model outputs, $f_i(\mathbf{x})$:

$$I_i(\mathbf{x}) = \frac{(z_i - E(f_i(\mathbf{x})))^2}{\text{var}(z_i - E(f_i(\mathbf{x}))} = \frac{(z_i - E(f_i(\mathbf{x})))^2}{\text{var}(e_i) + \text{var}(\epsilon_i) + \text{var}(f_i(\mathbf{x}))},$$

where $e_i$ is the observational error (difference between the observations and the real physical process) and $\epsilon_i$ is the model discrepancy (difference between the real physical process and the simulated computer model). For the model output and observations, large values of $I(\mathbf{x})$ imply that the predicted output of the model at $\mathbf{x}$ is far away from where we would expect it to be if $f_i(\mathbf{x})$ were consistent with the observations $z_i$. A threshold, $a$ is usually chosen to define the implausible input space. Here, the implausibility measure eliminates any input space that is unlikely to belong to one of the two distinct regions. Applying the full method, the input space then divides into these regions, $R_i$, such that the simulator output is smooth within each region, but discontinuous across boundaries. The method discussed in the paper is as follows: Sort inputs $\mathbf{x}$ into regions and take a sample from each region, $R_i$. Choose an initial subvector $\mathbf{z}_1$ from the observed data (across both regions). From the samples of each region, construct two emulators, $f_{(1)}$ and $f_{(2)}$. For each input, $\mathbf{x}$, from the subvector,

$\mathbf{z}_1$, two implausibility measures, $I_{(1)}(\mathbf{x})$ and $I_{(2)}(\mathbf{x})$ are calculated, where the model discrepancy, $\epsilon$, can differ for both regions. Two region subspaces of non-implausible input values, $C_{(1)}(\mathbf{z}_1)$ and $C_{(2)}(\mathbf{z}_1)$ associated with the respective emulators, are constructed, where an input is removed from the input space if it is not a member of either subspace. This process is continued by refocussing in waves within each of the two sub-regions, producing a decreasing sequence of subsets of input space.

This method can then be used to test which region a general input value belongs to. With each iteration of history matching, the next sample of points is taken from the subspace that is not yet ruled out from either of the regions, and a new emulator is formed using these points. This process continues until it is unclear which region the points will lie in, and hence the uncertainty is high near the border. Although this method has no smoothness assumption, it may still be difficult in higher dimensions. Hence, in both of these cases, I would prefer to use my method as it is more general.

## 2.7 The Design Problem

An important topic in uncertainty quantification is the design problem; if we could run the simulator again, where in the input space would be the optimum place to include more points? The design problem is discussed in detail by Sacks et al. (1989), who look at the criteria for choosing a design that predicts the response well at new points in the input space. One of these criteria is based on the mean squared error (MSE), by choosing the design that minimises the integrated MSE between the computer model output and observations. They show that this method is fairly easy to implement, and can be effective at reducing the squared error in prediction. Additionally, they look at choosing points that minimise the maximum MSE. This can be equally effective, but computationally complex for continuous regions. A Bayesian design scheme focuses on the minimisation of the expected posterior entropy which calculates the 'amount of information' in an experiment. There are many variations on this approach, all of which are explained by Sacks et al. (1989) and by Currin et al. (1991).

As discussed in Section 2.2.3, Ranjan et al. (2008) tackle this problem through the use of contour estimation. In their article, they work on improving the efficiency of estimating the boundary so that they can minimise the number of design points needed to obtain an accurate estimate. Initially, they prioritise points that are as close to being on the boundary as is possible, by choosing them such that they lie within a small neighbourhood of the current contour estimate. To improve on this, they develop an improvement function as stated in Equation (2.4), which is averaged over the uncertainty in the response surface. This is important for deciding whether it is better to choose points to explore the input space, or to choose points that are close to the contour. More recent work also includes that by Knudde et al. (2019).

Picheny et al. (2010) address the issue of designing experiments for metamodels that need to be accurate for particular level-sets of the response. Similar to Ranjan et al. (2008), they aim to construct a design such that the metamodel accurately estimates a contour. They describe similar approaches to space filling designs, maximum MSE and integrated MSE before presenting a variation called the 'weighted integrated mean squared error' criterion. This allows the user to put more weight on choosing points to improve the contour estimate, rather than reducing the overall variance of the model (as seen in just the integrated mean squared error method). Both this method, and that by Ranjan et al. (2008), give interesting details on how to adapt their designs to focus on improving the estimate of a boundary, which is definitely an important aspect to consider for the design problem considered here.

Lastly, Bect et al. (2012) work on estimating the volume of an excursion set of a function above a given threshold under a probability measure. They use what they call 'stepwise uncertainty reduction strategies' with an aim to improve the estimation of a probability of failure. Several possible methods are discussed which include analysing the Shannon entropy, minimising the utility as a loss function and minimising the quadratic loss function. Their chosen method is based on trying to minimise the probability of misclassification (the probability of predicting a point above the threshold when the true value is under). More recent work also includes that by Knudde et al. (2019).

All these would be valid approaches in a traditional setting, but for my model there is the key difference that the function being emulated is the labelling function, $\Lambda(.)$, which only takes values $l_1$ or $l_2$ for the corresponding regions. I also have a choice of whether it would be better to place points to find a better estimation of the boundary, or to create a more space filling design. If we are only interested in classifying the input space, then it is likely that we would only choose to place points that were close to the boundary. Having additional knowledge in the areas of the input space which are far away from the boundary is unlikely to give any extra confidence on where the boundary actually lies. Bearing in mind that if the number of regions is unknown, then if we are not careful in searching all areas of input space, we may find that we are falsely classifying. Alternatively, we may want to fit separate emulators to the regions after we have made the classification. In this scenario it is important that we both place points to give a better estimate of the boundary, and create a space filling design for both regions. For a space filling design, we would concentrate on including more points in sparse areas. I therefore do not want to use any design method that only bases its selection criteria on either the uncertainty or variance of the Gaussian process, as such methods will only choose to place points in the corners of the input space or in sparse regions.

One possible design method for my model makes use of the misclassification rate, and would choose additional points that maximise the change in misclassification rate across the whole input space. One way to do this is to sample several points in the input space from the latent Gaussian process. We would include each point at a time into the latent model, assuming that it is classified correctly, and calculate the misclassification rates as before. We then introduce the point into the design (and run the simulator or model function) that changes the maximum or average missclassification rate the most. Unfortunately, when trying to apply this method, there are problems with the leave-one-out validation used to calculate the misclassification rate. In the leave-one-out method, each point is left out in turn, the latent model is fitted to the remaining points, and this is used to predict the sign of the point left out, giving us a value for our misclassification. However,

this means that when the misclassification rate is calculated for a specific point, we are not including any information regarding that point, even though we are certain of its sign. So, when calculating the rate, we are ignoring the fact that we do in fact know the classification of that point, leading us to ignore the information for misclassification of the local points. Consequently, if we then attempt to pick the optimum point based on changing the maximum misclassification rate the most, we end up trying to choose points that are very close to points that we already know, and have included in our design. This method would ignore the local points, and so I can conclude that this is not an efficient way of tackling the design problem.

One way of trying to solve the above problem is to add a penalty so that we do not end up choosing points that are too close together. For new point, $\mathbf{x}'$, this can take the form: $\textit{missclassification} + \lambda \min \parallel \mathbf{x} - \mathbf{x}' \parallel$, to ensure that we add more weight to the points that are furthest away. The addition of this penalty then assures that our design is more space filling. When implementing this process, we could choose a larger value of $\lambda$ to begin with (to be more space filling), then gradually reduce $\lambda$ to focus on the accuracy of the classifier. Although adding this penalty is feasible, I still find that the method becomes computationally expensive, and so I feel that it would be better to tackle this problem from another angle.

Some alternatives to this are, for example, to maximise the change in total area of misclassification rate by integrating or taking the sum over the input space, as well as fitting emulators to the misclassification rate. I could also use a second set of points as a separate test or verification set (similar to Bastos and O'Hagan (2009)). This would mean that we would have one set of data which we use to fit the main model, and a separate set of data which we use to validate the latent Gaussian process, and produce an independent misclassification rate. We would then avoid the problems of ignoring local points in the misclassification, so that we can properly measure how much we mislabel things, and still produce a clean estimate. We could also continue to use a distance penalty to control how space filling our design is at each step. I believe that this would be a successful approach, but also believe that it may not be very practical due to the large amount of initial data required. Since

all of these suggestions still have computational problems, I instead use a different method, which I will detail below.

Due to the computational difficulties in using the misclassification rate to base my design problem on, I instead look for solutions that use the latent model itself. In Figure 2.9, I am able to plot the probability of being classified into region 1 for the entire input space. This is generated by taking samples of the latent GP, and predicting the classification for a grid of points covering the input space. I therefore propose that we use this uncertainty information to judge where the next best place would be to choose a point to run the simulator at. I decide that anywhere the probability gets close to 50% is the ideal place to choose a new point, since the latent model is most unsure where the boundary is. The implementation of this method focuses on the exploitation side of the design problem. I have discussed above how it is important to also consider the exploration aspect of the design problem. We could, for example, choose all points (from a selection) that are in the range 45-55% of being classified into region 1, and then choose the point which is the most space filling. To increase the exploration side further, we could change the size of the interval that we are concentrating on. For example, choosing the most space filling point from the interval 40-60% will explore the input space more than just looking at 45-55%. As before, I would prefer for the method to be more space filling when placing the initial extra points, and then for later points to focus on improving the classification. By making sure we are exploring all space before we start to concentrate on improving the estimate of the boundary, we ensure that we go into some of the regions where we are falsely confident in our classification estimate. To maximise the explore feature, we would just include the entire input space in the interval before choosing the most space filling point. For future research, it may be advantageous to find a way of optimising the range interval at each step. To decide which point is the most space filling, I look at the Euclidean distance from each point to each other point, and choose the point that maximises the minimum distance. See Section 2.8 below for an example.

## 2.8 Another Example in 2 Dimensions

I apply my method to a two-dimensional example (provided by T. Santner after a private communication), with test function:

$$f(\mathbf{x}) = \begin{cases} \infty & \text{if } x_1^2 + x_2^2 \leq c_1^2 \\ \frac{\exp -(a'\mathbf{x}+\mathbf{x}'Q\mathbf{x})}{(x_1^2+x_2^2-c_1^2)} & \text{if } c_1^2 \leq x_1^2 + x_2^2 \leq c_2^2 \\ -\infty & \text{if } x_1^2 + x_2^2 \geq c_2^2, \end{cases}$$

where,

$$a = [3, 5], \quad Q = \begin{pmatrix} 2 & 1.5 \\ 1.5 & 4 \end{pmatrix}, \quad c_1^2 = 0.25^2, \quad c_2^2 = 0.75^2 \, .$$

This function is plotted in Figure 2.13. The space between the two circles is $R_1$ and

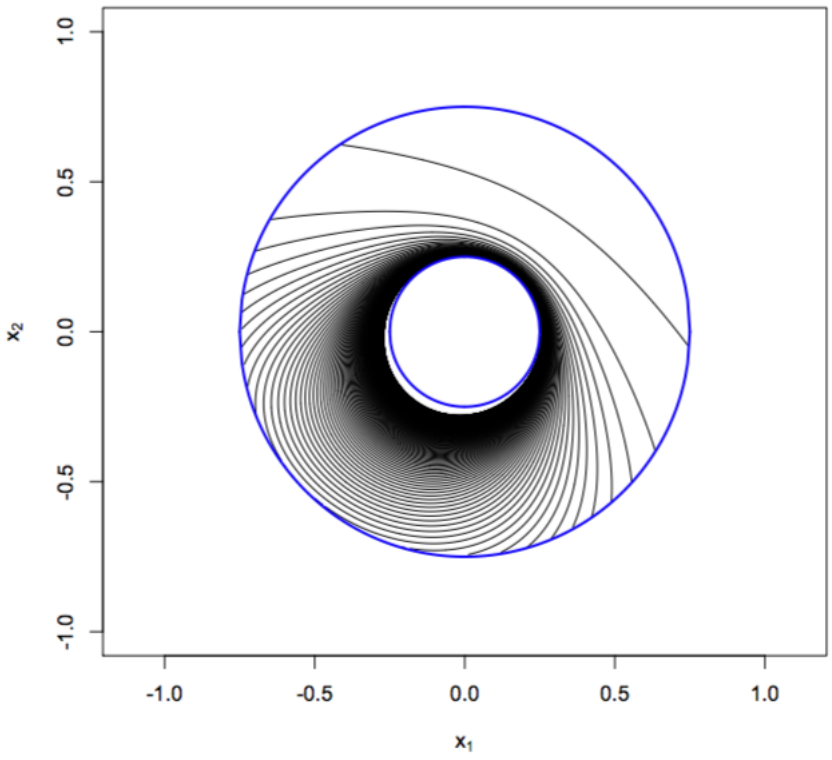



Fig. 2.13 2 dimensional example with two regions. $R_1$ lies within the two circles and $R_2$ is the remaining input space. The contours show the function, $f$, for various values of $x_1$ and $x_2$.

the remainder is $R_2$, both over the input space $[-1.25, 1.25]^2$. The output function to the model, $f$, is only valid for $R_1$, and we ignore the difference between $-\infty$ and

$\infty$ in Equation 2.8. Hence, the true labelling function, $\Lambda$, becomes:

$$\Lambda(x_1, x_2) = \begin{cases} l_1 & \text{if } \; 0.25^2 \leq x_1^2 + x_2^2 \leq 0.75^2 \\ l_2 & \text{if } \; x_1^2 + x_2^2 < 0.25^2 \text{ OR } x_1^2 + x_2^2 > 0.75^2. \end{cases}$$

I have used a 2-d maximin Latin hypercube to select 50 data points, $(x_1, x_2) \in D$, where they are given class labels ($l_1$ if $(x_1, x_2) \in R_1$ and $l_2$ if $(x_1, x_2) \in R_2$). These are shown by the purple and orange points in Figure 2.14 along with the hard boundary (red). The labelling classification after applying my method is also shown in the plot with uncertainty as the background colour. As in the previous example, the light blue areas represent a high probability of being labelled $l_1$ and the dark blue areas show high probability of being labelled $l_2$. The largest areas of uncertainty correspond to the areas where the classification method performed the poorest. Figure 2.15 shows two draws from the latent GP. The plot on the left is fairly accurate to the truth, but it is particularly interesting to note that the doughnut shape in the right plot is no longer fully connected. This is likely to be due to the lack of information in that area of input space.

Overall, my method is estimating the regions well with only a few larger deviations in the upper left and right sections of the doughnut region. This is likely to be caused by the lack of information in these areas. Due to the more complicated shape, I chose to fit a constant prior mean function. This has proved to be successful since no areas have been misclassified in the far corners of the input space. Alternatively, a quartic polynomial could have been used for the prior mean function, but, as discussed in Section 2.4, I do not recommend using a polynomial with greater complexity than a quadratic (unless there is a sufficient quantity of data). Adding the quartic prior would involve estimating 8 additional parameters.

Two input points of particular interest are those at the bottom of the larger circle; they are labelled in different regions but are very close together. In this area, the latent Gaussian process must change sign quickly but has been able to without any complications.

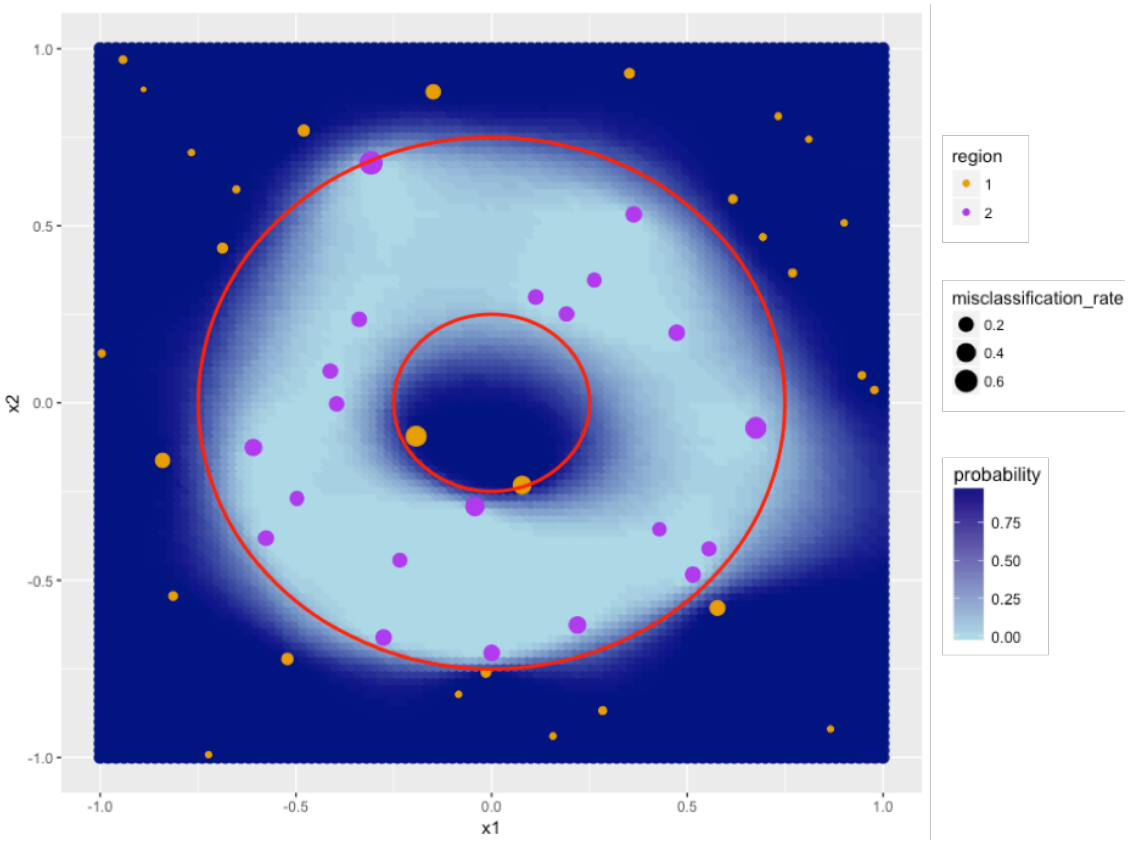



Fig. 2.14 Estimated regions for the 2-d example shown in Figure 2.13. Initial data points are displayed (orange - Region 1 and purple - Region 2), with the actual region boundaries shown in red. Uncertainty on the estimate is included where light blue areas correspond to high probability of being classified into $R_1$ and dark blue areas correspond to high probability of being classified into $R_2$. A misclassification rate is also shown.

A misclassification rate is also included in Figure 2.14, where the points are more likely to missclassify in $R_1$ (between rings). This is likely to be due to a higher proportion of points being in $R_2$ and so the majority of the latent process is negative, making it more likely for areas to be classified into $R_2$. This is supported by the constant mean function estimated to be $-2.25$.

The larger misclassification rates (compared to those in Figure 2.9) can also be attributed to the use of a constant mean function. It becomes a lot more uncertain without the directional force of a higher order polynomial. Although it appears that the corners misclassify very infrequently, when I observe the underlying latent Gaussian process (not shown), $\eta$, I can see that it is in fact starting to curve up in the corners towards the constant mean value. But since the mean value is $-2.25$, there will not be a misclassification in this case.

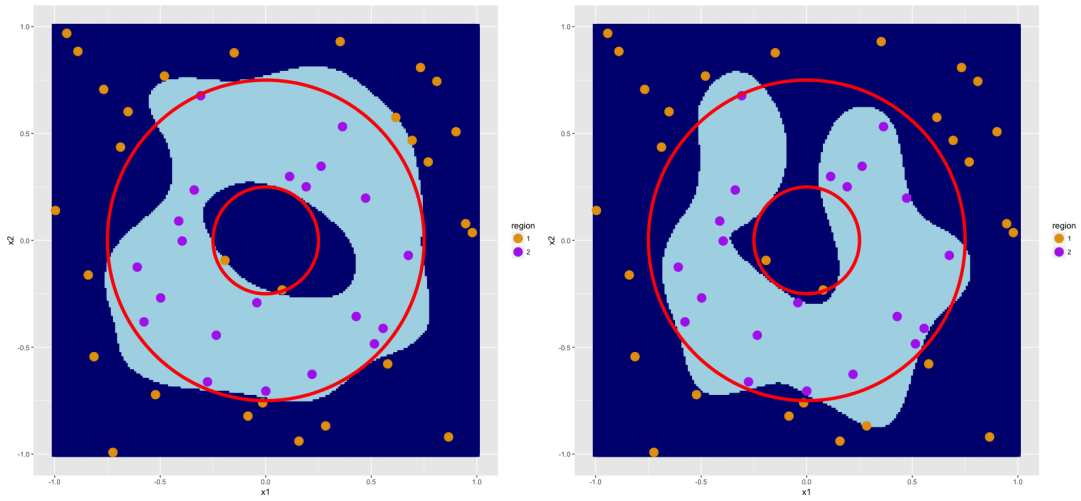



Fig. 2.15 Two different draws from the 2 dimensional example with two regions. $R_1$ lies within the two circles and $R_2$ is the remaining input space. The dark blue and light blue regions correspond to areas being classified into $R_1$ and $R_2$ respectively.

### 2.8.1 Design

The design problem is an important aspect of uncertainty quantification, as outlined in Section 2.7. Figure 2.16 displays the output of applying the design method in Section 2.7 to the doughnut example shown here. To apply this, I have chosen to run the simulator at 20 extra points (one at a time) and then observe the difference the extra points have made to the overall classification of the two regions. As said previously, my method is based on choosing regions of points where the classifier is most uncertain, where there is close to an equal chance for the point to be sorted into either of the regions. Then from these points, I chose the one that is most space filling based on a maximin criterion (Sacks et al., 1989). I am able to change the design to either be more space filling or to concentrate on estimating the border by altering the area of points that we look at to choose the most space filling points. If we were focusing on the border, we might choose points with 45-55% probability of being classified into region 1, but if we were trying to be more space filling, we might choose the area of points that has 30-70% chance of being classified into region 1.

For the first 14 points, I chose to alternate between selecting points to be very space filling and improving the boundary. Therefore, I alternated between choosing to pick the most space filling point from the whole input space and from the top 40% of points that have a probability of being classified into region one close to 50%.

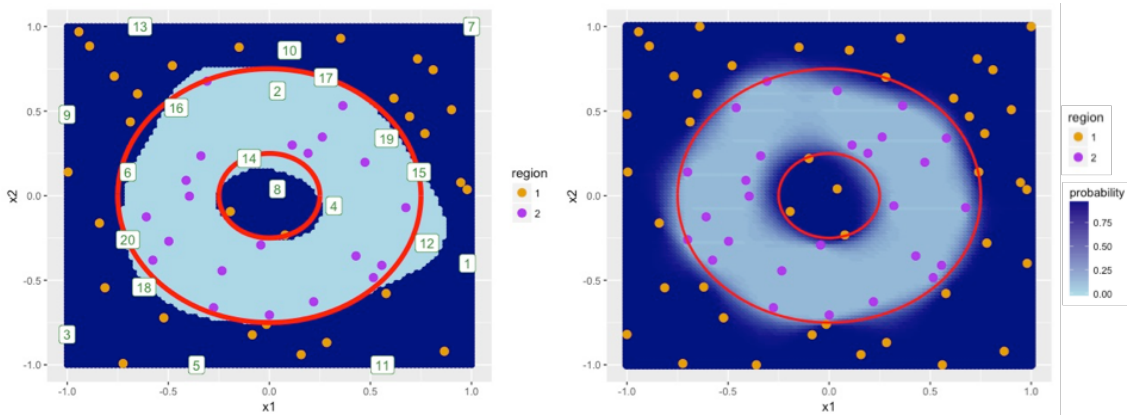



Fig. 2.16 Plots to show design implemented to the example in Figure 2.14. Left: Original estimate for the classification of the input space with extra points added. Points are labelled to the order they would be added in the design. Right: New estimate for the classification with the extra 20 input points included. Light blue shows high probability of being classified into region 1 and dark blue show high probability of being classified into region 2.

I felt this was important to make sure that there weren't any areas of the input space where I was wrongly predicting the classification. For the remaining 6 points, I decided to just focus on improving the classification by choosing the top 1-5% of points that have a probability of being classified into region 1 close to 50%. From the plot, you can tell this has proven to be successful since the last 7 points have been placed very close to the actual boundary between regions. This is ideal since I have still not given the model any knowledge of where the actual border is and the accuracy of our classification is based on how well we can estimate the boundary between regions.

The right plot in Figure 2.16 shows the output of the model when I have included all 20 new points into the initial data. As before, the light blue areas of input space show high probability of being sorted into region 1, and the dark blue areas show high probability of being sorted into region 2. Comparing this to the original estimate in Figure 2.14, we can see that there is definitely an improvement in our classification as well as a reduction in uncertainty near the borders. Before, there was a large bulge in the estimate towards the bottom right of the input space. However, with the new data points included, this is no longer the case and the estimate is hugging the border much more tightly. The new points have also greatly improved the estimate of the centre of the doughnut with it appearing much more accurate to the circular

shape. Previously, the classification had a 8% misclassification rate, however this is reduced to just 4% after the extra design points have been included.

## 2.9 Application

My motivating example has been supplied by Voliotis et al. (2018) where the subject is the reproductive system in mammals, particularly how this is controlled by connections between the brain, the pituitary gland, and the gonads. There are particular neurones in the brain that secrete a specific hormone known as the gonadotrophin-releasing hormone (GnRH). These are vital in regulating gametogenesis and ovulation. Signals are made by the pituitary gland which then simulate the gonads for this cycle to start. One of the regulators of the GnRH neurone is neuropeptide kisspeptin, of which two are located within areas of the hypothalamus (the arcuate nucleus (ARC) and the proptical area). Other research suggests that one of these areas (ARC) is the location of the GnRH pulse regulator of which the core are neurones (ARC kisspeptin or KNDy) that secretes two neuropeptides: neurokinin B (NKB) and dynorphin (Dyn). The objective of the model presented is to understand the role of NKB and the firing rate of these neuropeptides on the regulation of GnRH, and subsequentially in controlling reproduction. To do this, the model identifies the population of the KNDy neurones where the GnRH pulse regulator is said to be found. The model consists of a set of coupled ordinary differential equations (ODEs) to describe the dynamics of $m$ synaptically connected KNDy neurones. There are several fixed parameters including the concentration of Dyn, rates at which Dyn and NKB are lost and those that describe the characteristic timescale for Dyn and NKB. The variables are the concentration of NKB secreted at the synaptic ends and the firing rate, measured in spikes/min. Using the population of KNDy neurones is shown to be critical for GnRH pulsatile dynamics and that this can stimulate GnRH secretion. Analysing the output of this model shows that the population can behave as a bistable switch so that the firing rate is either high or low. Hence, this causes us to have a system with two distinct solutions, and is an example of the type of system that I wish to model. This bistable system is coupled

with a negative feedback leading to sustained oscillations that drive the secretion of GnRH hormones that are involved in reproduction. Being able to model the system and locate the areas of low and high firing rates means that, not only can we aid predictions on the reproduction rate, but we can also have a better understanding of the specific input parameters that are associated with high rates of reproduction.

The inputs are NKB concentration and firing rate, where I create a Latin hypercube over the input space of $[0.1, 0.2] \times [10, 200]$. The choice was made to transform the data to $[0, 1]^2$ for computational simplicity. The system is bimodal with two labels, where I have 20 initial points with known region classification. I apply the labelling function, $\Lambda$ where there are 5 points labelled $l_1$ in $R_1$, and 15 labelled $l_2$ in $R_2$ (as seen in Figures 2.17 and 2.18). The true function, $\Lambda$ and resultant boundary are not known in this example.

One of the most important choices to be made in this example was the form of the prior mean on the latent Gaussian process. I chose a linear prior based on consultation with the expert (M. Voliotis) of the system and examination of the initial points (yellow and purple shown in Figure 2.17). The output of the predicted region boundary is shown in Figure 2.17 along with uncertainty. In general, the solution classifies as expected in most areas, where the area between the regions is the most uncertain. We would therefore expect the true boundary between $R_1$ and $R_2$ to be almost a straight line, with potential to curve at either of the ends of the input space. This uncertainty is down to lack of information in the area, but since the area of uncertainty is not too large, I have greater confidence in my estimation. The model is computationally expensive to run, hence we cannot run the model extensively to compare the boundary estimate with the truth. Figure 2.18 shows two draws from the latent GP and confirms that there is more uncertainty in the lower half of the input space close to the boundary. They both capture a similar linear trend, with the plot on the right having more curvature. Misclassification is also shown through the size of the points where we can notice that it is more likely to misclassify points near the boundary.

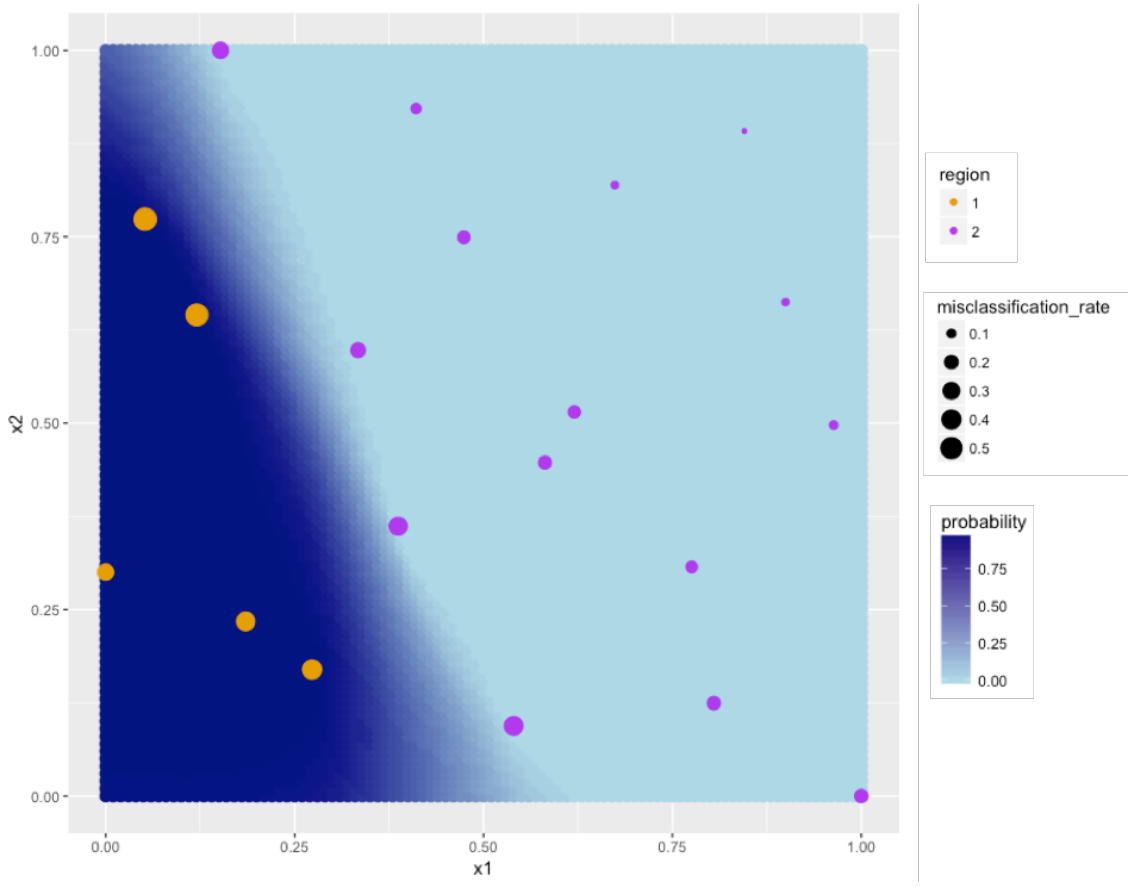



Fig. 2.17 2 dimensional example looking at the effects of hormone release on mammal reproduction, where the system has two regions of high and low rates of hormone release. Initial points are displayed (orange - $R_1$ and purple - $R_2$), with predicted region classification and uncertainty. Dark blue areas correspond to high probability of being classified into $R_1$ and light blue areas correspond to high probability of being classified into $R_2$. A misclassification rate is also shown.

## 2.10 Discussion

I have developed a new method for classifying models or simulators into two distinct regions. My method includes correlation through a distance metric and it can be applied to a broad range of applications where outputs to the model are not necessarily quantitative. I use aspects of classification from Nickisch and Rasmussen (2008) in the form of class labelling and incorporate Gaussian process emulation. To ensure that correlation between data points is included, a latent variable modelled as a Gaussian process is used to structure the two output solutions using the assigned class labelling. The latent Gaussian process is estimated using Metropolis Hastings MCMC with distinct prior specifications. As a form of model validation, I have

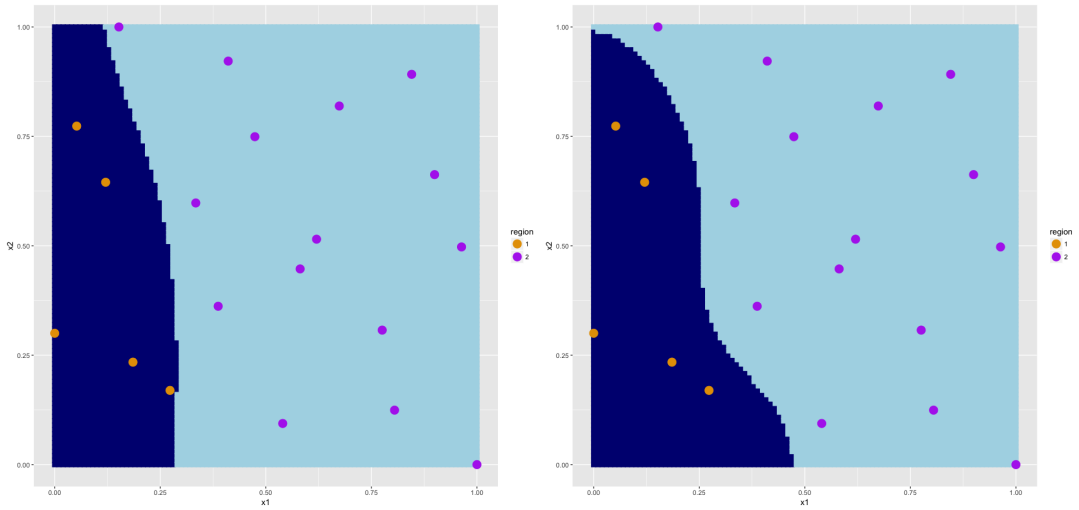



Fig. 2.18 Two different draws from the 2 dimensional application with two regions. Initial inputs are shown in yellow and purple with the dark blue and light blue regions corresponding to areas being classified into $R_1$ and $R_2$ respectively.

calculated a misclassification rate which is based on a leave-one-out cross-validation. I have also included details on how I would tackle the design problem.

I feel that this method will be applicable to a wide range of applications across many disciplines including computer science, climate science and biology. My main motivating example is based on assessing reproduction rates in mammals (Voliotis et al., 2018). I have successfully modelled this bimodal system, where it can be used for class prediction for other input points with estimates of uncertainty included. It would be interesting to try my method on more real life data sets to see how well it performs for more intricate output regions.

Comparisons have also been made with logistic regression and Voronoi tessellation. There are several other approaches to problems that are similar to the one presented here that would be interesting to make further comparisons with. For example Pope et al. (2018), have developed a method for modelling spatial processes with heterogeneity or discontinuities by using a combination of Voronoi tessellation and Gaussian processes. As well as this, I would like to reach deeper into machine learning classification methods to see if there are any similar comparisons or ideas on how I can improve my method.

There are some other important extensions to the work presented in this chapter. One would be to now expand the method to cope with situations when there are

more than two output solution classes. Although I outlined how I would like to tackle this problem, I would like to improve on this and apply it to some real life data. Ideally I would like to adapt our model to cope with more solution regions, however this is likely to be challenging due to the positive and negative layout that I have chosen.

There is also room for research in areas of experimental design where I can improve the accuracy of the classification and boundary estimation with limited initial data. This has been quite a difficult problem due to the fact I am applying the Gaussian process to the labelling function and not the actual model output itself. This means that a lot of current design methods are no longer applicable. I have touched on the direction that I would like to go in, but I would like to continue this research to make sure that this is the most efficient method. I would also like to improve on my method by creating an optimiser (similar to the improvement function used by Ranjan et al. (2008)) that would indicate whether we should be choosing points to be more space filling or to improve the boundary estimate. I would also like to improve the efficiency of the design by having a feature where we can choose multiple new points at a time.

There are also obvious extensions in improving the efficiency of the computer code. For example, I could change the MCMC to Hamiltonian MCMC to speed up the calculations.

# Chapter 3

# De Bruijn Graphs

## 3.1 Introduction

The aim of this chapter is to investigate a correlated Bernoulli process that improves on the independent drawing problems that are known to occur when using classification methods such as logistic regression (Chang et al., 2016; Diggle et al., 1998). Logistic regression models the probability of being in one of two regions and, when classifying new points, it is common practice to take draws from an independent Bernoulli distribution. From these draws, we then choose to arbitrarily use 0 to represent one of the regions, and 1 to represent the other. As shown in Figure 1.3 in Chapter 1, taking independent draws with probability determined by the logistic regression means that all of the distance correlation that we initially had is lost. Hence, we end up with many misclassifications, and we cannot easily produce a clean cut boundary between regions. Although we can take many independent Bernoulli draws and average them to create a smooth boundary, this would not be a complete solution as we would still have to assign a threshold to find the two regions.

Therefore, the aim of this chapter is to produce a Bernoulli process where correlation is incorporated when selecting draws or samples. This novel process must have a high correlation between points that are close together and a low correlation for points that are far apart. In a one dimensional scenario, this would correspond to being able add structure to a sequence of 0's and 1's so that we can force when like symbols cluster together instead of appearing uncorrelated. When classifying,

we will then hopefully observe a clean cut boundary between regions (as shown in Chapters 1 and 2), instead of having frequent misclassifications. As we saw in Chapter 1, the multivariate Bernoulli distribution introduced by Teugels (1990) was promising, as it can produce probability distributions for whole sequences of 0's and 1's for a set number of individual Bernoulli trials, but his method can quickly get very complicated for long sequences. Hence, I seek to improve on his method in a way that incorporates the between data correlations into calculating the probability of occurrence of a given sequence.

From alternative literature, Tallis (1962) looked into finding a generalised multinomial distribution that invokes correlation in discrete data. He first defines a random variable, $X$, that takes the values $0, 1, 2, ..., k$ (which is slightly more general than the proposed method here, which has two values, 0 and 1). He also defines a probability generating function for the joint probabilities, $P(X_1 = a, X_2 = b, X_3 = c, ...) = \alpha_{abc...}$ $(a, b, c = 0, 1, 2, ..., k)$. A generating function is often easier to work with than the actual distribution itself (Wilf, 1994). The generating function of Tallis (1962) is defined in terms of a parameter, $\rho$, which is the correlation coefficient between variable pairs in $X$. Later on in the paper, he states that $E[X]$ must remain constant throughout, which is something that we may not always want to hold for the method which is the focus of this chapter. The work was continued in Tallis (1964), where the author determines how to estimate the distribution of a sum of non-independent multinomial variates.

Although the method presented in Tallis (1962) gives the required attention to the correlation between points that I require, there is still extra work that would be necessary before it could be applied to the problem at hand. We would first have to ensure that the correlation between variable pairs is based on a distance measure, so that there is a higher correlation between points that are close together. We may also want to further control the spread of the correlation, so that it is not only between pairs. For example, we could include three-fold correlations, or higher, into the model. This controlling of the spread of correlation is similar to the correlation length scale parameter involved in Gaussian processes. Alternatively, if we wanted

to continue working only with correlation pairs, then we could say that we only care about the nearest neighbours; i.e., conditional on second order terms, the third order terms (and higher) do not give us any additional information.

We would then also need a method for inference and simulation so that, given a set of parameters, we could produce sequences of 0's and 1's that follow the patterns produced from the region classifications. Tallis (1962) shows how to estimate the relevant parameters using maximum likelihood, but we would still need to consider how to apply this to the given problem to make classification predictions, given logistic regression.

Alternatively, Dai et al. (2013) consider how to use a multivariate Bernoulli distribution to estimate the structure of graphs with binary nodes, somewhat similarly to the structures used in Teugels (1990). The advantages of this method, as compared with that of Tallis (1962), is that it is able to take account of higher order interactions as well as pairwise interactions. This allows much more structure to be incorporated into the model. They also consider the multivariate Bernoulli distribution in the framework of the exponential family, where the binary nodes are described as random variables formed to allow pairwise relationships in terms of the edges. This is where correlation is included into their model: variables are conditionally independent if the associated nodes are not linked by an edge.

For example, Figure 3.1 shows a graph with the set of variables $\{N1, N2, N3, N4, N5\}$ displayed as the nodes of the graph. The edges between nodes show us which variable pairs have correlation and which ones are conditionally independent. $N1$ is directly correlated to variables $N2$, $N3$ and $N5$. Although $N1$ is conditional on $N2$, N1 and N4 are conditionally independent when conditioned on N2. $N4$ is only dependent on $N2$ and independent from all other variables. Nodes $N1$, $N3$ and $N5$ form a cycle in the graph. Additionally, arrows can be placed on the edges to show the direction of the dependency.

It is interesting to view the correlated Bernoulli problem using a graphical structure as it gives a clear indication of the dependence between variables that are linked on the graph, and the independence between variables that are not. One of

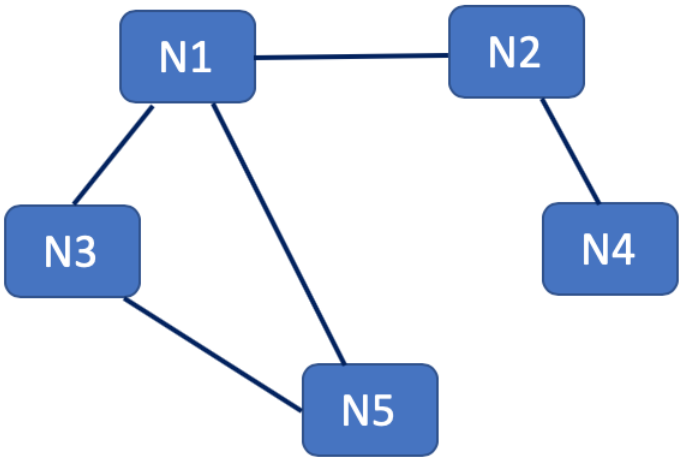



Fig. 3.1 Example of a graph with nodes representing the variables, $\{N1, N2, N3, N4, N5\}$. Edges drawn between nodes represent the dependencies between variables.

the largest problems with both the method from Dai et al. (2013), and that seen in Teugels (1990), is the complexity of the distribution, and the number of parameters to be estimated. For example, for sequences of 0's and 1's of length two, we already require three parameters. As the sequences under consideration get longer, the number of parameters explodes, making the inference problem infeasible for the types of problems considered here.

Leisch (1998) focuses on generating multivariate binary random variables. The method he proposes is to transform normally distributed random variates to binary values via componentwise thresholding to include the correlation between variates. The pairwise relations between variables can be written as a covariance matrix, or by specifying pairwise probabilities. He also addresses the problem specified above with the methods introduced in Dai et al. (2013) and Teugels (1990), where the number of parameters to be estimated explodes when looking at long chains of multivariate Bernoulli trials. Leisch (1998) suggests converting the probabilities with a lower-dimensional parametrisation so that the number of parameters to be estimated is equivalent to the number of 0's and 1's in the chain, $n$. This is a substantial reduction from the previously estimated value of $2^n$.

Although this method seems interesting and there exists an R package (bindata), I did not take this any further since the paper was never published and the author did not continue the work.

Due to the described problems with these methods, I looked for other possible solutions, and came across the notion of de Bruijn graphs. They appeared to capture a lot of what was needed, including higher order interactions, and even a graph structure similar to that seen in Dai et al. (2013). The rest of this chapter outlines what de Bruijn graphs are, and how I wish to use them to produce a correlated Bernoulli process.

## 3.2 De Bruijn Graphs

**Definition** (de Bruijn Graphs): De Bruijn Graphs (De Bruijn, 1946; Fredricksen, 1992; Golomb, 1967; Good, 1946) are directed graphs consisting of overlapping sequences of symbols: given a set of $s$ symbols, $V = \{v_1, ..., v_s\}$, the vertices or nodes of the graph consist of all the possible sequences of $V$. Each graph has $s^m$ vertices, where $m$ is the length of each possible sequence given the set of symbols, $V$. The possible nodes are as follows:

$$V^m = \{(v_1, ..., v_1, v_1), (v_1, ..., v_1, v_2), ..., (v_1, ..., v_1, v_s), (v_1, ..., v_2, v_1), ..., (v_s, ..., v_s, v_s)\}.$$

Edges in de Bruijn graphs are drawn between node pairs in such a way that the connected nodes have overlaps of $m - 1$ nodes. An edge is created by removing the first symbol from the node sequence, and then adding a new symbol to the end of the sequence from $V$. Thus, from each vertex, $(v_1, ..., v_m) \in V^m$, there is an edge to vertex $(v_2, ..., v_m, v) \in V^m$ for every $v \in V$. There are exactly $s$ directed edges going into each node and $s$ directed edges going out from each node. I name the symbols, $v$, the 'letters' of the de Bruijn graph, and the sequence of these letters at each node a de Bruijn 'word' of length $m$.

By travelling along a random walk through the de Bruijn graph, we are able to create chains of letters that are all dependent on the $m$ letters that come before (Fredricksen, 1992). Hence, by altering $m$, we are able to change the dependence structure of the de Bruijn graph. A de Bruijn sequence of order $m$ on a size-$s$ set of letters, $V$, is a cyclic sequence in which every possible length-$m$ string on $V$ occurs exactly once as a substring. Such a sequence is denoted by $B(s, m)$ and has length $s \times m$, which is also the number of distinct substrings of length $m$ on $V$ (Golomb, 1967).

Figure 3.2 shows an example of an $m = 2$ de Bruijn graph consisting of the set of letters $V = \{\text{A,B}\}$. All of the possible length-two sequences are $V^2 = \{\text{AA,AB,BA,BB}\}$. These sequences make up the four nodes of the graph, where there are two directed edges both coming out and going in to each node. The end of the initial sequence is the same as the start of the next sequence that each edge is attached to.

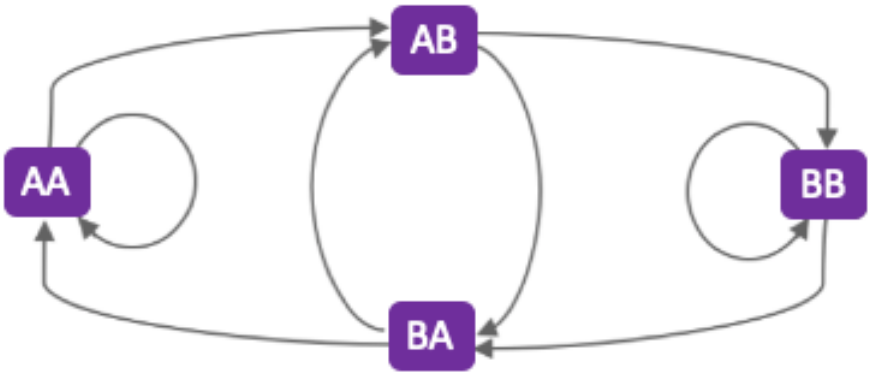



Fig. 3.2 Example of a length two de Bruijn graph with the letters A and B

De Bruijn graphs are used in a number of applications including genome sequencing (Compeau et al., 2017), and the mathematics of juggling (Ayyer et al., 2015). In genome sequencing, de Bruijn graphs are used to assemble whole genome sequences given small collections of nucleotides, which are substrings of the required genome. The bases A, G, C and T that constitute the sequences of DNA are the possible de Bruijn letters and, given a set length $m$, we can decompose each of the substrings into overlapping sequences of length $m$. Once this has been done, any sequences of

length $m$ that match up across the different substrings can be put together to start forming the full genome sequence.

For example, say we have the two substrings, TCAAGGTTA and GGTTACGT, and we want to identify the whole genome sequence that they come from. If we were using a de Bruijn graph of length $m = 4$, then we could decompose the two substrings into the sets of sequences; {TCAA, CAAG, AAGG, AGGT, GGTT, GTTA} and {GGTT, GTTA, TTAC, TACG, ACGT} respectively. This gives us all the sequences of length 4 from both of the substrings, where each one is created by shifting down the line one letter at a time. Noticing that the two sequences, GGTT and GTTA overlap in the two substrings means that the resulting sequence is TCAAGGTTACGT.

Additionally, in the mathematics of juggling, de Bruijn graphs are used to model the patterns balls make when being juggled. For example, imagine you have 4 slots for balls (one in each hand and two in the air), and you can have either ball or not ball at each of the slots. A ball can go from your hand to the air, but it can not go from one hand to the other if there is already a ball there - hence there needs to be some conditioning structure in the modelling that controls where each ball is allowed to go at each time.

Given a de Bruijn graph such as that in Figure 3.2, one can assign a value to each of the directed edges to show the probability of transitioning from each node to the next. We can thus immediately see that there is a connection to Markov chains (see below) (Billingsley, 1961; Ching and Ng, 2006; Feller, 1950; Isaacson and Madsen, 1976). The next possible node to transition to depends only on the current node and a set of probabilistic rules. This choice structure introduces the required correlation.

A Markov chain is a stochastic system where transitions occur from one state to the next according to probabilistic rules. At each time step $t$, the probability of transitioning to the next state is solely dependent on the current state and time step. Markov chains are often described as 'memoryless'. This means that the current position depends only on the previous state, but not any further back, nor any future states. This is known as the Markov property.

**Definition** (The Markov Property): For random variables, $X_0, X_1, ...$, any positive integer, $t$ and possible states, $i_0, i_1, ..., i_t$, a stochastic process forms a Markov chain if $P(X_t = i_t | X_{t-1} = i_{t-1}, X_{t-2} = i_{t-2}, ..., X_0 = i_0) = P(X_t = i_t | X_{t-1} = i_{t-1})$.

If we compare the Markov property to the conditional dependence in de Bruijn graphs, we notice that there is a crucial difference. I make a key definition here that de Bruijn graphs have a Markov property on the de Bruijn word and not the letter. The current word is dependent on only the previous word in the sequence, and no other. This means that we can create far more structure than if it were simply the letters that were Markov. For example, looking at Figure 3.2, the word AB can follow either the word AA or BA, and we can assign different probability values to both of these possible transitions. AB is thus dependent on the whole word that comes before it, and not just the single letter A. The letters can instead be viewed as having a latent Markov property, as they are where we observe the Markov structure in the sequences, but it is the words that actually create the Markov chains.

We can change how many letters we are dependent on by altering the length of the word, $m$. The length of each word in the de Bruijn graph tells us how spread the correlation is over nearby data points (which is somewhat equivalent to a length scale since it is the distance over which things are correlated). If we have a length $m = 2$ de Bruijn graph, we are dependent on two letters back, and if we have a length $m = 3$ de Bruijn graph then we are dependent on three letters back. If $m = 1$, then the model collapses down to be classically Markov (Markov on the letter), and if we have a zero length de Bruijn graph, this is just equivalent to independent Bernoulli trials.

Since a de Bruijn graph is Markov on the word, the transition probabilities give us the the probability of transitioning from word to word. I denote this $p_i^j$ to mean the probability of transitioning from the word $i$ to the word $j$. Like Markov chains, all of the transition probabilities for the de Bruijn graph can be written into a matrix, $T$. Each element of the matrix, $T$, at time step, $t$, is given by $T_{ij} = P(X_{t+1} = j | X_t = i) = p_i^j$. The corresponding transition matrix for the example

in Figure 3.2 is as follows:

$$T = \begin{pmatrix} p_{AA}^{AA} & p_{AA}^{AB} & 0 & 0 \\ 0 & 0 & p_{AB}^{BA} & p_{AB}^{BB} \\ p_{BA}^{AA} & p_{BA}^{AB} & 0 & 0 \\ 0 & 0 & p_{BB}^{BA} & p_{BB}^{BB} \end{pmatrix} = \begin{pmatrix} 1 - p_{AA}^{AB} & p_{AA}^{AB} & 0 & 0 \\ 0 & 0 & 1 - p_{AB}^{BB} & p_{AB}^{BB} \\ 1 - p_{BA}^{AB} & p_{BA}^{AB} & 0 & 0 \\ 0 & 0 & 1 - p_{BB}^{BB} & p_{BB}^{BB} \end{pmatrix},$$

where $p_{AB}^{BA}$ is the probability of transitioning from the word $AB$ to the word $BA$. Due to a conservation of probability property, each row of the transition matrix is a probability vector, and so every row in the matrix must sum to one; i.e. $\sum_{j=1}^{2^m} T_{ij} = 1$ for all $i$. Hence, as there are $s = 2$ letters in this example, there are only two edges coming out of each node of the graph and we can state $p_{AA}^{AA} = 1 - p_{AA}^{AB}$. This reduces the number of transition probabilities by half.

### 3.2.1 Further Markov Properties

Since de Bruijn graphs are Markov on the word, we are able to use some of the properties of Markov chains to state properties for de Bruijn graphs. Before we look into these, there are a few important definitions (Norris, 1997; Ross, 2014).

**Definition** (Irreducibility): A Markov chain is irreducible if it is possible to get from every state to every other state with positive probability.

**Definition** (Aperiodicity): An irreducible Markov chain is said to be aperiodic if for some $t \geq 0$ and state $j$:

$$P(X_t = j | X_0 = j) > 0 \qquad \text{and} \qquad P(X_{t+1} = j | X_0 = j) > 0$$

The stationary (or ergodic) distribution of a Markov chain (Isaacson and Madsen, 1976; Jones and Smith, 2001) tells us the proportion of overall time spent at each state or word. It describes how the chain will be distributed among the states after it has been run for a long amount of time. The state of the system at time $t$ is given by $\pi^{(t)} = (\pi(1)^{(t)}, ..., \pi(m)^{(t)})$ so that each component $\pi(i)^{(t)}$ indicates the probability of being at each word, $i$, at time $t$.

**Definition** (Stationary Distribution): The stationary distribution of a state $j \in S$ from an irreducible, persistent, aperiodic Markov chain is defined as:

$$\pi(j) = \lim_{t\to\infty} \; p_i^{j(t)},$$

where $p_i^{j(t)} = P[X_{k+t} = j | X_k = j]$. The $\pi$'s must also satisfy the following:

$$\pi(j) > 0, \qquad \sum_{j\in S} \pi(j) = 1 \quad \text{and} \quad \pi(j) = \sum_{i\in S} \pi(i) p_i^j,$$

The stationary distribution is hence found when $\pi^{(t)}$ remains unchanged as $t$ gets larger. Since $\pi(j) = \lim_{n\to\infty} \; p_i^{j(n)}$ and $p_i^{j(n+1)} = \sum_k p_i^{k(n)} p_k^j$ for states $k \in S$, we can find the stationary distribution by letting $n \to \infty$. This gives:

$$\pi(j) = \sum_k \pi(k) p_k^j,$$
$$\implies \quad \pi = \pi T,$$

for transition matrix $T$. I will denote the stationary distribution as $\pi$ for the remainder of this thesis.

Providing the Markov chain on the word is stationary (discussed below), we can calculate the stationary distribution by powering up the transition matrix as follows (Isaacson and Madsen, 1976):

$$P(X_t = i) = \sum_j P(X_t = i | X_0 = j) P(X_0 = j)$$
$$\implies \quad \pi^{(t)} = T^t \pi^{(0)},$$

for time step $t$ and states $i, j$. Let $t \to \infty$, then we are left with:

$$\pi = \lim_{t\to\infty} T^t \pi^{(0)},$$

which gives us a row vector for the stationary distribution. For these Markov chains, the marginal probabilities are irrespective of the starting state, $\pi^{(0)}$, and so for any starting chain the steady state remains the same.

The stationary distribution of a Markov chain can also be calculated by looking at the eigenvalues and eigenvectors of the transition matrix, $T$ (Feller, 1950; Isaacson and Madsen, 1976). Considering the equation, $\pi T = \pi$, we can see that this looks very similar to the column vector equation for eigenvalues and eigenvectors, $Av = \lambda v$ (where A is a matrix and v is the eigenvector associated to the eigenvalues $\lambda$ of A). By letting $\lambda = 1$ (and $A = T$, $v = \pi$) and taking a transpose we have the following:

$$\begin{aligned} & (T\pi)^T = \pi^T \\ \implies \quad & \pi^T T^T = \pi^T. \end{aligned}$$

We can see that for the transposed matrix, $T^T$, the eigenvector that is associated to the eigenvalue $\lambda = 1$ is $\pi$, which is the stationary distribution.

The eigenvalues and eigenvectors can also make powering up the transition matrix $T$ computationally easier. This is down to using the eigendecomposition of $T$, which is shown by Isaacson and Madsen (1976). Let $L$ be a matrix where the rows of $L$ are the left eigenvectors of $T$. Let $\Lambda$ be a diagonal matrix where the diagonal consists of the eigenvalues of $T$, and all other elements are zero. Since $LT = \Lambda L$ and $L$ is invertible, we get:

$$L^{-1}LT = T = L^{-1}\Lambda L.$$

Taking powers of $T$, we find that $T^2 = L^{-1}\Lambda L L^{-1}\Lambda L = L^{-1}\Lambda^2 L$, which gives the following general result:

$$T^t = L^{-1}\Lambda^t L,$$

where $\Lambda^t$ is a diagonal matrix with diagonal elements, $\lambda_1^t, \lambda_2^t, \ldots, \lambda_m^t$.

Not only can we find the stationary (or ergodic) distribution from the eigenvalues and eigenvectors, but we can also find other important properties. One of these is the convergence rate of the Markov chain, and hence the convergence rate of the de Bruijn graph towards the stationary distribution of the words (Jones and Smith, 2001; Ross, 2014).

**Definition** (Convergence Rate): The convergence rate of a Markov chain is the speed in which $\pi^{(t)}$ approaches the stationary distribution, $\pi$, as $t \to \infty$.

If $\lambda = 1$ is an eigenvalue of $T$, then for the remaining eigenvalues, $|\lambda| < 1$ (as proved by Rosenthal (1995) and Isaacson and Madsen (1976)). Number the eigenvalues such that $1 = |\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_m|$ and let $x^T = \sum_{i=1}^{m} a_i l_i$ for $a_i \in \mathcal{R}$, where $l_i$ is the $i$th row of $L$.

$$\begin{aligned}
\pi^{(t)} &= xT^t \\
&= xL\Lambda^t L^{-1} \\
&= (a_1 l_1^T + a_2 l_2^T + \ldots + a_m l_m^T) L\Lambda L^{-1} \\
&= a_1\lambda_1^t l_1^T + a_2\lambda_2^t l_2^T + \ldots + a_m\lambda_m^t l_m^T \\
&= \lambda_1^t \left\{ a_1 l_1^T + a_2 \left(\frac{\lambda_2}{\lambda_1}\right)^t l_2^T + \ldots + a_m \left(\frac{\lambda_m}{\lambda_1}\right)^t l_m^T \right\}.
\end{aligned}$$

Since the stationary distribution is given by $l_1$ and $\lambda_1 = 1$ , we find that $\lambda_2$ is the dominant term. Therefore the second eigenvalue tells us how fast the Markov chain converges to its stationary distribution. For a fast convergence rate, we want eigenvalues close to one.

We can then state the expected first return time for a particular state or word, $i$ (Ross, 2014).

**Definition** (Expected Return Time): For a Markov chain with stationary distribution $\pi$, the expected first return time $R_i$ for state $i$ is given by:

$$R_i = \frac{1}{\pi(i)}.$$

This is the associated long run time of the word $i$.

The mean passage time to get from word $i$ to word $j$ is then given by $\frac{n_{ii} - n_{ij}}{\pi(i)}$ where $n_{ij}$ is an element from the matrix $N = (I - T - W)^{-1}$ with $I$ being the identity matrix and $W$ being a matrix whose rows are the stationary distribution $\pi$.

### 3.2.2 Non-Stationary Markov Chains and Further Connections

The type of Markov chains we have been discussing for de Bruijn graphs are so far not dependent on the time step, $t$ (Feller, 1950). These are said to be stationary or homogeneous in time.

**Definition** (Stationary Markov Chain): A Markov chain is stationary if the probability of going from one state to another is independent of the time step. For all states $i$ and $j$, we can state that $P(X_t = i|X_{t-1} = j) = P(X_1 = i|X_0 = j)$ for any time $t = 0, 1, 2, ....$

Alternatively, there exist Markov chains where this doesn't necessarily have to be the case, and the probabilities of transitioning can change dependent on both the current state and time. These are known as non-stationary Markov chains where we are able to change the probabilities for transitioning to certain words dependent on the current time step. For the example in Figure 3.2, the transition matrix is now as follows:

$$T^{(t)} = \begin{pmatrix} 1 - p_{AA}^{AB(t)} & p_{AA}^{AB(t)} & 0 & 0 \\ 0 & 0 & 1 - p_{AB}^{BB(t)} & p_{AB}^{BB(t)} \\ 1 - p_{BA}^{AB(t)} & p_{BA}^{AB(t)} & 0 & 0 \\ 0 & 0 & 1 - p_{BB}^{BB(t)} & p_{BB}^{BB(t)} \end{pmatrix},$$

for $t = 0, 1, 2, ....$

If we are dealing with a non-stationary de Bruijn graph, many of the Markov properties discussed above are no longer applicable (Isaacson and Madsen, 1976; Ross, 2014). For example, a stationary distribution or steady-state does not typically exist for Markov chains with non-stationary transition probabilities. An important exception are periodic Markov chains where the transition matrix repeats after a set number of time steps. If the transition matrix repeats after $q$ time steps such that, $T^{(t+q)} = T^{(t)}$, then the stationary distribution is as follows:

$$\pi(j) = \lim_{n\to\infty} p_i^{j(nq+t)},$$

where $\pi(j) = \pi^{(0)}T^{(1)} \dots T^{(j-1)}$ with $\pi^{(0)} = \pi^{(0)}R$ and $R = T^{(1)}T^{(2)} \dots T^{(q)}$. The main disadvantage with this (and most calculations for non-stationary Markov chains) is that it is not analytically tractable and an approximation must be found.

For de Bruijn graphs, if transitioning from word to word is dependent on the time step, this allows us to change the pattern of letters in the chains at different times. We would also be able to determine when certain states are more likely to

be visited, which introduces additional structure for the determination of when like letters will clump together. Although it would be ideal to apply all of my future work in this thesis to non-stationary Markov methods it is very challenging. Hence, I will mostly work with stationary Markov chains, with the aim that I can extend my work to non-stationary cases at a later date.

De Bruijn graphs as Markov chains have many connections to alternative methods in the literature. One of the most obvious examples are higher order Markov chains (Ching and Ng, 2006; Raftery, 1985; Wang et al., 2013). The main difference between higher-order Markov chains and the first-order Markov chains we have considered so far is that the state probability distribution of the chain at the present time depends on the $m$ previous time steps; i.e., the state of the Markov chain at time $t$ is dependent on the states at times $t-1, t-2, ..., t-m+1$.

One simple way of defining a higher order Markov chain is to rewrite the states so that we combine the previous $m$ states into one new state. If we do this at each time step, then we have the ability to combine the correlation from all $m$ previous time steps into a simple first-order Markov chain. As you might have noticed, this is exactly what a de Bruijn graph does, where the higher-order states are the letters and the transformed first-order states are the words. Therefore, de Bruijn graphs are equivalent to higher-order Markov chains where we can simply change the order or spread of correlation through the word length $m$. Both have parameters of $\mathrm{O}(s^m(s-1))$, given there are $s$ states in the model.

An alternative higher-order Markov chain was developed by Raftery (1985) with the aim to reduce the number of parameters. These higher-order Markov chains have a Markov dependency on a linear combination of previous individual letters and are defined in the definition below. Similarly to above, the states of the Markov chain are the individual letters, and we are looking at the probability of transitioning to the next letter dependent on the past $m$ letters in the chain.

**Definition** (Higher-order Markov Chain): If we have a sequence, $\{X_t : t \in \{1, 2, ...\}\}$, then the higher-order Markov model takes the form:

$$P(X_t = i_0 | X_{t-1} = i_1, ..., X_{t-m} = i_m) = \sum_{j=1}^{m} \lambda_j q_{i_0 i_j},$$

where, $\lambda_1 + ... + \lambda_m = 1$, and $Q = q_{ik}$ is a transition matrix such that $0 \leq \sum_{j=1}^{m} \lambda_j q_{ik_j} \leq 1$ for $i, k_1, ..., k_m = 1, ..., t$. This shows that the probability of the current state is dependent on a linear combination of the past $m$ states.

As we increase the number of time steps that the current letter is dependent on we add an extra parameter into the model, so that the total number of parameters is of $\mathrm{O}(ms^2)$ if the Markov chain has $s$ states (Ching et al., 2008). Although this number of parameters is larger than that of de Bruijn graphs ($\mathrm{O}(s^m(s-1))$), I prefer the simplicity of using the first-order Markov chains, and so proceed with de Bruijn graphs for the remainder of this thesis. It is my aim to produce methodology for a correlated Bernoulli process using de Bruijn graphs, with the hope that I can reduce the number of parameters at a later date. Thus, although I will not be considering higher-order Markov chains any further, they may be important for future work in parameter reduction.

A further connection to de Bruijn graphs which we shall discuss is that with group theory, where previous work includes that by Hauge and Mykkeltveit (1996) and Rhodes et al. (2017). A de Bruijn graph can be thought of in terms of permutations, as the individual words create an ordering for the full sequence. A directed cycle in group theory is simply a finite sequence of elements that can be generated by running a random walk on the de Bruijn graph. Cycles are created by repeatedly applying a permutation to the elements in a set, which are equivalent to the words (or nodes) of the de Bruijn graph. Taking each word in turn, we apply the permutation (by running through the Markov chain), and determine the next word in the sequence. We can then form separate cycles (or sets) from various subsets of the nodes in these graphs. In the shift register literature (Golomb, 1967), there are also techniques to calculate the longest cycle of 0's and 1's that contain all words without any repeats. There are many existing theorems that can be applied to the de Bruijn structures,

such as cyclic decomposition of Markov chains (Isaacson and Madsen, 1976; Sonin, 2008). A Markov process can thus be decomposed into a collection of directed cycles with positive weights which are proportional to the probability of travelling through a cycle in a random walk. In graph theory, cyclic decomposition is taken to be the partitioning of the edges of the graph and so gives the same interpretation as for Markov chains.

Although I do not go into the details of group theory in this thesis, it is a very vast and important topic for future work as it may be able to offer simplifications, or allow generalisations to higher dimensions. As well as group theory, there are also links to graph theory, where I will discuss de Bruijn graphs expressed as trees in Section 3.3.3.

The final property that I am interested in is the expected run length of certain letters of the de Bruijn graph, which I investigate in Chapter 4.

## 3.3 Towards a Correlated Bernoulli Process

Now that I have outlined what de Bruijn graphs are, I will now move on to show how they can be used for the problems considered in this thesis. The aim of this work is to be able to produce sequences of 0's and 1's, for which same letters cluster together. By incorporating correlation between variables using the de Bruijn structure, I aim to produce an improvement on drawing independently from a Bernoulli distribution.

An interesting approach to using de Bruijn graphs within the context of a correlated Bernoulli process is outlined by Ayyer and Strehl (2011), who look at producing a stationary distribution for a de Bruijn process. They begin with a very similar structure to the one proposed here, by viewing the de Bruijn graphs as generalised Markov chains with associated transition probabilities. They let the set of letters be $\{1, 2, ..., n\}$ and proceed by decomposing de Bruijn sequences into blocks, such that they are a repetition of a single letter. For example, an $a - block$ is a word which is the repetition of the single letter $a$ $k$ times in a row, displayed as $a^k$. Every sequence of letters has a unique decomposition into these blocks of maximal length. A weight is then assigned to each node made up of these blocks, which is

determined by the letter and number of repetitions. Through the use of a Kirchhoff matrix, the resulting transition matrix is found. The de Bruijn process is then a continuous time Markov chain for the de Bruijn graph with specified weights. The stationary distribution of the probabilities of given sequences are found conditional on the given weights.

In one example they refer to a de Bruijn-Bernoulli process, which refers to when the associated weights cause the stationary distribution to be a Bernoulli measure. Although this sounds similar to the goal considered here, one major flaw is that the probabilities of being in each Markov state are independent to each other.

The disadvantages of the work of Ayyer and Strehl (2011) mainly lie in the objectives. The authors focus on trying to find the stationary distribution of sequences from the underlying Markov chain along with any properties associated with it. In contrast, I am more interested in learning about the transitions, and what a random walk through the de Bruijn graph would look like. Further, their transition probabilities are treated as variables in time, whereas I initially want to keep them constant to be able to use stationary Markov properties. Their paper is also very complex and I think it would greatly benefit from a new set of notation, simplifying the majority of definitions. Therefore, although I do not take any of the ideas further in this thesis, it is still interesting to note the connections and layouts of notation and de Bruijn graph definitions.

Rhodes et al. (2017) focus on random walks on de Bruijn graphs, and present a definition for a de Bruijn-Bernoulli process. Whilst quite similar to the goal here, a deeper reading indicates that it does not have the required form. First, they have used the term ‘de Bruijn-Bernoulli process’ to refer to attaching an independent probability to all of the transition edges on the graph, such that there is now a corresponding probability to adding any of the letters in the de Bruijn graph (and not the word). This has the effect of removing the dependence of the current state on the one before in the Markov chain, hence removing all the correlation that I want to retain. Therefore, I do not look into this further and instead create my own correlated Bernoulli process.

To link de Bruijn graphs with this idea of a correlated Bernoulli process, I will be dealing with the set of $s = 2$ letters, $V = \{0, 1\}$. This is similar to Fredricksen (1992) who introduces a binary de Bruijn graph. The order of the de Bruijn graph is then the length of each word that makes up the nodes of the graph, where these words will be length $m$ sequences of 0's and 1's. The de Bruijn graphs of lengths $m = 2$ and $m = 3$ for $V = \{0, 1\}$ are shown in Figure 3.3. The possible words are shown by the blue and orange nodes. Since the graphs are made up of two letters (0 and 1), there are two edges coming in and out of each node so that there are a total of $2^m$ nodes and $2^{m+1}$ edges. The middle plot emphasizes the fact that the $m + 1$ word length graph can be created by placing an appropriate node on each of the existing edges in the $m$ word length graph and then adding in the corresponding edges. Hence, for the $m + 1$ length de Bruijn graph there are an extra $2^m$ nodes and $2^{m+1}$ edges.

A de Bruijn graph is Markov on the word. Hence, the nodes of the de Bruijn graph are the states of a Markov chain, where the edges are the transition probabilities for either adding a 0 or 1 to the sequence depending on the current word. As a reminder, the notation, $p_i^j$, represents the transition probability of going from state (word) $i$ to state (word) $j$; e.g., for word length $m = 3$, $p_{010}^{100}$ is the probability of going from the word 010 to the word 100. This is the same as saying a 0 is added to the sequence 010 to create the sequence 0100. By running through the de Bruijn graph, it is then possible to create long chains of 0's and 1's.

The transition probabilities can be written in terms of a transition matrix. For example, for a de Bruijn graph of word length 2 (top graph in Figure 3.3), the possible words are: 00, 01, 10 and 11. The corresponding transition matrix is formed as follows:

$$T = \begin{pmatrix} 1 - p_{00}^{01} & p_{00}^{01} & 0 & 0 \\ 0 & 0 & 1 - p_{01}^{11} & p_{01}^{11} \\ 1 - p_{10}^{01} & p_{10}^{01} & 0 & 0 \\ 0 & 0 & 1 - p_{11}^{11} & p_{11}^{11} \end{pmatrix},$$

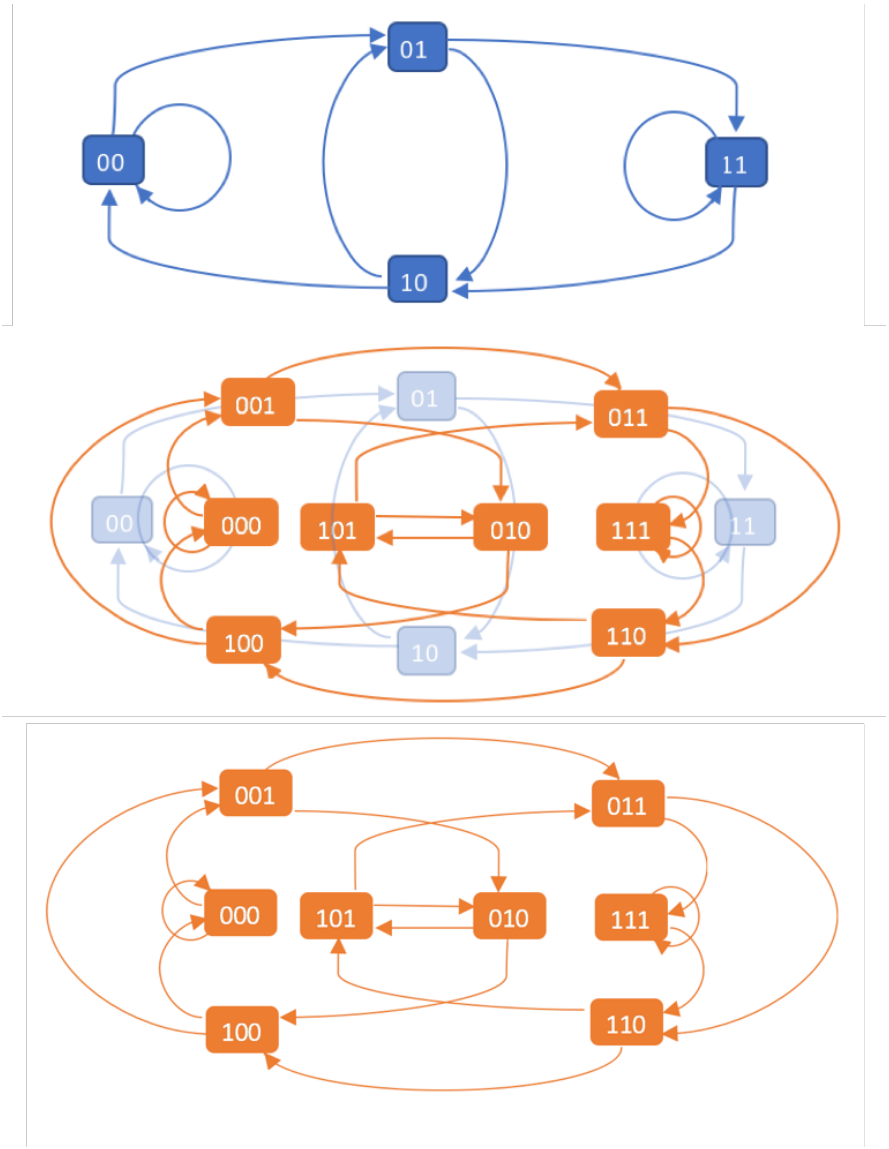



Fig. 3.3 Examples of length 2 and 3 de Bruijn graphs with two letters: 0 and 1.

where $p_{00}^{00} = 1 - p_{00}^{01}$ due to conservation of probability. As the word length increases, we end up with more transition probabilities and an increasingly large transition matrix. For word length $m$ there are $2^{m+1}$ different transition probabilities (same as the number of edges in the graph).

I can now incorporate this to formally define the new process, which I name the de Bruijn process. This is a correlated Bernoulli process, and has the following properties. I note that the set of letters does not necessarily have to be $V = \{0, 1\}$, but this is used for the rest of work in this thesis.

**Definition** (de Bruijn Process): The de Bruijn process is a process to produce sequences of 'letters' from the set, $V = \{0, 1\}$, where correlation is included through a de Bruijn graph structure with length $m$ 'words'. There is defined to be a Markov

property on the de Bruijn words but not on the letters such that for time step, $t$:

$$P(X_t = i_t | X_{t-1} = i_{t-1}, X_{t-2} = i_{t-2}, ..., X_0 = i_0) = P(X_t = i_t | X_{t-1} = i_{t-1}) \\ = p_{i_{t-1}}^{i_t},$$

for random variables, $X$, and where $p_{i_{t-1}}^{i_t}$ is the probability of transitioning from the word $i_{t-1}$ to word $i_t$.

The main aim for developing this de Bruijn process is to be able to produce chains of 0's and 1's so that we get blocks of letters, and avoid getting spikes of misclassified values that occur when making predictions using a classification method or logistic regression. The word length and associated transition probabilities are where we can add a certain amount of 'stickiness' into the model so that we are able to control how clumped together the letters will be. For example, if we let the probabilities be: $\{p_{00}^{01} = 0.1, p_{01}^{11} = 0.9, p_{10}^{01} = 0.1, p_{11}^{11} = 0.9\}$ in a length $m = 2$ de Bruijn process, this will ensure that there is a high level of stickiness for both 0's and 1's, avoiding changes between values. Equivalently, we can make the model very 'anti-sticky' by choosing the transition probabilities that retain the current letter to be small. It is a combination of both the transition probabilities and the word length that gives us the vast about of possible structure. As we increase the word length, this generates a larger number of intricate transition probabilities where we can set values to give us the exact de Bruijn process we are looking for.

As before, I distinguish the marginal probabilities, $\pi$, from the transition probabilities, $p$, to avoid confusion. I state $\pi(\{i\})$ when referring to $i$ as a letter, and $\pi(i)$ when referring to $i$ as a word. If the Markov chain on the word is stationary, then the marginal probabilities (for both words and letters) can be calculated using the methods described in Section 3.2. The probabilities of getting a 0 or 1 will vary as you go through (dependent on the current word), but the marginal probabilities of getting 0's and 1's will be constant over time.

Since it is the transition probabilities that control the structures of the letters, there are relationships between the marginal probabilities and the transition probabilities which we can express in the following equations. For $m = 2$ we know the following

relationships are true:

$$\begin{aligned}\pi(\{1\}) &= p_{00}^{01}\pi(00) + p_{01}^{11}\pi(01) + p_{10}^{01}\pi(10) + p_{11}^{11}\pi(11),\\ \pi(\{0\}) &= p_{00}^{00}\pi(00) + p_{01}^{10}\pi(01) + p_{10}^{00}\pi(10) + p_{11}^{10}\pi(11),\end{aligned} \tag{3.1}$$

where $\pi(\{0\}) + \pi(\{1\}) = 1$. These expressions are derived from the law of total probability that states that $\pi(\{j\}) = \sum_n P(j|i_n)\pi(i_n)$, so that the probability of a letter (0 or 1) is the weighted average of all the possible words that could generate that letter. In other words, we average over all of the possible starting words to get the probability of obtaining a single 0 or 1. We are able to obtain the $\pi's$ from the transitions, $p$, but we are not able to obtain all of the transitions from just knowing the marginals, $\pi$. There needs to be some constraints to make the transition probabilities identifiable.

For the above example (where $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\} = \{0.1, 0.9, 0.1, 0.9\}$), the marginal probabilities for the words are: $\{\pi(00), \pi(01), \pi(10), \pi(11)\} = \{0.45, 0.05, 0.05, 0.45\}$, and the marginal probabilities for the letters are: $\pi(\{0\}) == 0.5$ and $\pi(\{1\}) = 0.5$. There is a 50% chance of getting a 0 or a 1, but by including the de Bruijn graph structure into the Markov chains, we are forcing correlation to be included and so the 0's and 1's will appear in clustered blocks (see examples in Figures 3.4 and 3.6). If we remove the de Bruijn structure and generate a sequence of independent random Bernoulli trials, we would observe the same proportions of each letter, but they would no longer be clustered in blocks.

We can expand Equation (3.1) to be applicable for any word length, $m$, as follows:

$$\pi(\{1\}) = \sum_{i=0}^{2^m-1} p_i^{(2i+1) \bmod 2^m}\pi(i),$$

where $\pi(\{0\}) + \pi(\{1\}) = 1$. This is written in the same structure as for the word length 2 case in Equation (3.1), but to simplify the notation and to apply to a general word length, we can write the words in terms of the decimal representation of the

binary values, e.g. for word length $m = 3$ the words translate as follows:

$$\begin{aligned} 000 &\rightarrow 0 \\ 001 &\rightarrow 1 \\ 010 &\rightarrow 2 \\ 011 &\rightarrow 3 \\ 100 &\rightarrow 4 \\ 101 &\rightarrow 5 \\ 110 &\rightarrow 6 \\ 111 &\rightarrow 7 \end{aligned}$$

This form is repeatedly used for the remainder of this thesis and is formally written as, $\sum_{i=1}^{m} k_i \, 2^{i-1}$, where $k_i \in \{0, 1\}$ is each letter in the word; e.g. for the word 010, $\sum_{i=1}^{m} k_i \, 2^{i-1} = k_1 2^0 + k_2 2^1 + k_3 2^2 = 0 \cdot 1 + 1 \cdot 2 + 0 \cdot 4 = 2$.

All of the calculations in this and the following chapters are based on sequences and run lengths of 1's. However, since there are only two possible letters, there are some symmetries, and we can easily convert this to be applicable to 0's.

### 3.3.1 Examples

Figures 3.4, 3.5, 3.6, 3.7 and 3.8 present some examples of how the transition probabilities and word length effect the chains of 0's and 1's that are produced when running through the corresponding de Bruijn process. To generate a sample from the de Bruijn processes, we run through the transition matrix to generate long runs of 0's and 1's. It is also currently assumed that the starting value is unknown. We simulate for a sufficient amount of time to be in steady state, taking a large enough burn-in to ensure that this does not affect any results. This is equivalent to a random walk on a de Bruijn graph.

Figure 3.4 shows the effects of altering the transition probabilities. Each panel gives one of four examples of running a word length $m = 2$ de Bruijn process for $n = 200$ time steps. The marginal probabilities are kept the same ($\pi(\{0\}) = \pi(\{1\}) = 0.5$) so that we can make a better comparison of the spread of letters.

We would expect that like letters become far more sticky when the probabilities for remaining at the same letter are kept close to one. If the transition probabilities are all kept at 0.5, then we would expect the opposite, which would be the equivalent of a series of random Bernoulli trials. The transition probabilities set for each $m = 2$ example are $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\}$, where for the four examples (from top to bottom) these parameters take the values $\{0.5, 0.5, 0.5, 0.5\}$ , $\{0.25, 0.75, 0.25, 0.75\}$ , $\{0.1, 0.9, 0.1, 0.9\}$ and $\{0.05, 0.95, 0.05, 0.95\}$ respectively. The first (top) example behaves as expected, where the distribution of the 0's and 1's appears fairly random. The 0's and 1's in the second and third plots then show increasingly sticky behaviour, to the point that the final (fourth) plot shows a large region where the de Bruijn process has favoured 1's over 0's. Note that there are not equal numbers of 0's and 1's in the final plot because the average run lengths are becoming much larger and it is harder to find a length 200 window when the marginals are equal. If I let this graph run forever, then over time we would be left with equal numbers of 0's and 1's.

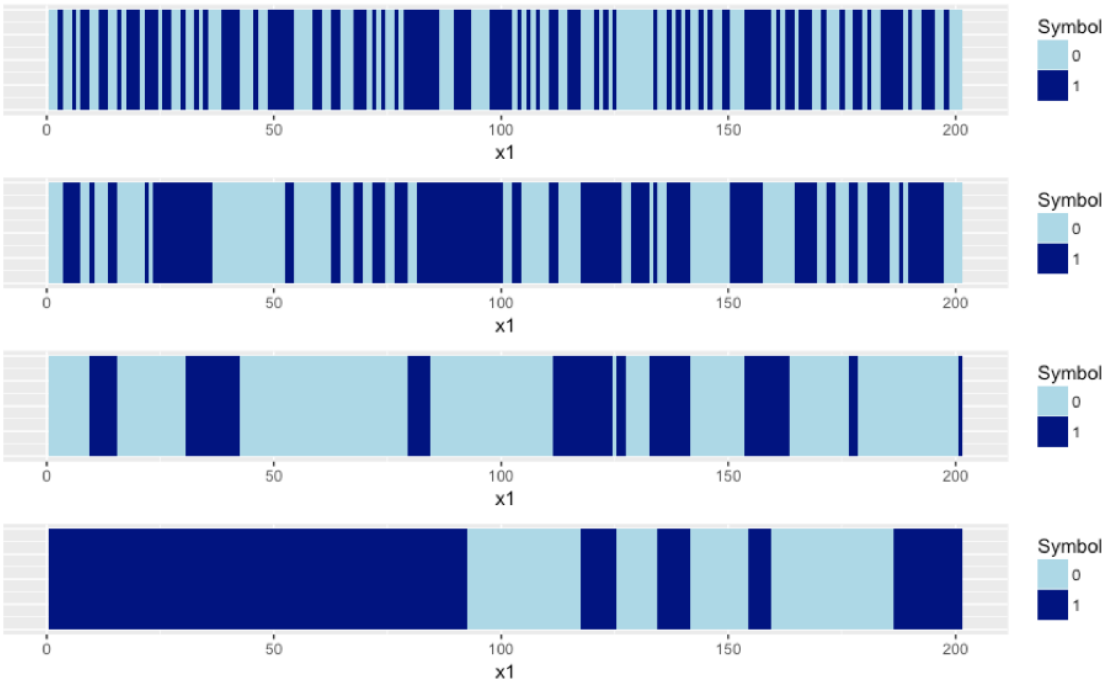



Fig. 3.4 Four samples from de Bruijn processes with letters 0 and 1 to show the effects of changing the transition probabilities. From top to bottom, the transition probabilities, $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\}$, are: $\{0.5, 0.5, 0.5, 0.5\}$ , $\{0.25, 0.75, 0.25, 0.75\}$ , $\{0.1, 0.9, 0.1, 0.9\}$ and $\{0.05, 0.95, 0.05, 0.95\}$.

The corresponding run length histograms for these examples are shown in Figure 3.5. A run length is defined as follows:

**Definition** (Run Length): A run length is defined as the number of consecutive 1's (or 0's) in a row bounded by a 0 (or 1) at both ends.

The plots in Figure 3.5 show the distributions of these run lengths for given sequences of 0's and 1's. The histograms confirm the results indicated in Figure 3.4, by showing that not only are there far fewer shorter run lengths when the letter stickiness increases, but also that the more extreme run lengths are far larger. We still see some shorter run lengths in the very sticky de Bruijn process plots, but there are far less, which is to be expected in a stochastic process. There is also far less variation in the top plot: for the majority of the time we see run lengths of length one or two.

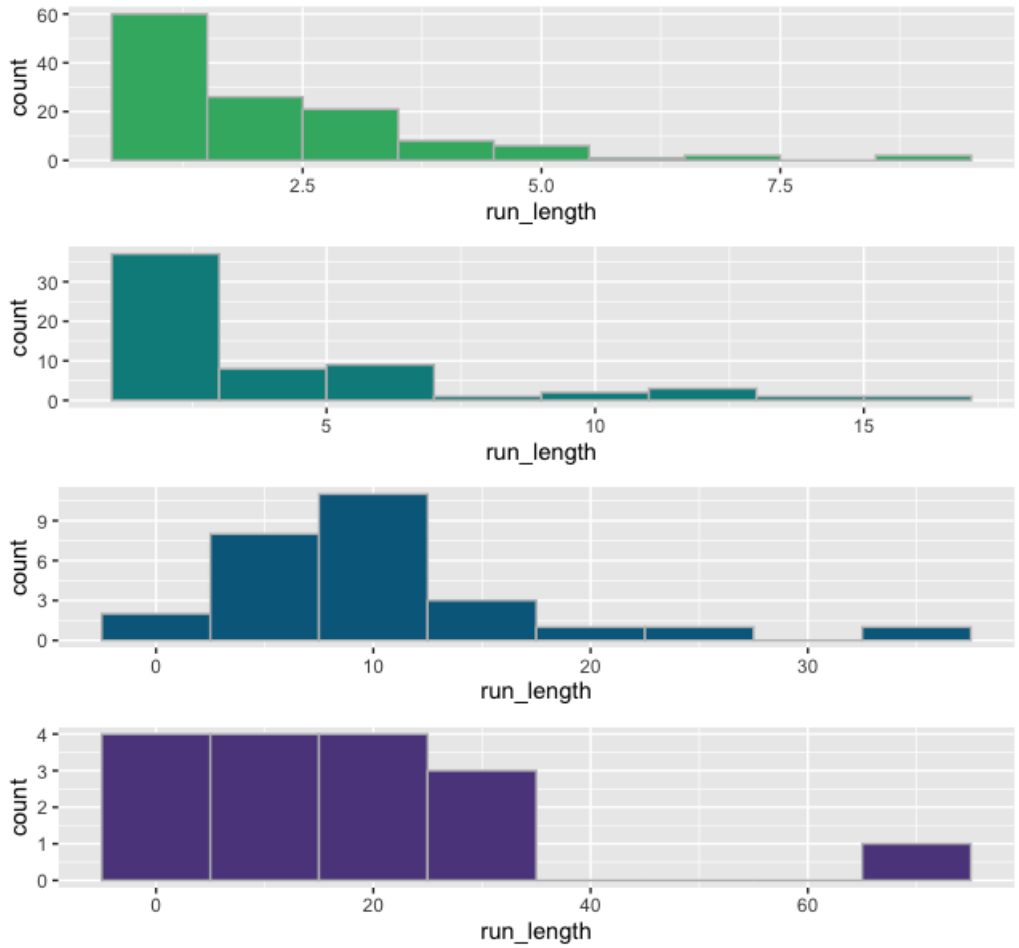



Fig. 3.5 Corresponding histograms showing the distributions of run lengths of 1's from the de Bruijn process examples in Figure 3.4. The run lengths are shown to increase as the de Bruijn processes become more sticky to 1's.

As a measure of stickiness of the de Bruijn process, we can also look at the average run length of 1's (or 0's) in a corresponding chain. This is discussed in

greater detail in Chapter 4, so I will just state the run lengths for the examples in Figure 3.4 here. The average run lengths of 1's are 1.95, 4.10, 9.47 and 19.60 respectively for each of the four simulations of length 200. These results are what we would expect, given that the average run lengths are getting increasingly longer as the stickiness of the de Bruijn process is increased. It is interesting to note that although there is not a huge difference between the probabilities, $\{0.1, 0.9, 0.1, 0.9\}$ and $\{0.05, 0.95, 0.05, 0.95\}$, there is a substantial difference between their average run lengths. During a run of 1's, there is a 10% chance for the next letter to be a 0 for the $\{0.1, 0.9, 0.1, 0.9\}$ de Bruijn process, but a 5% chance for the next letter to be a 0 for the $\{0.05, 0.95, 0.05, 0.95\}$ chain.

As stated previously, it is also possible to alter the transition probabilities to create a structure that is anti-sticky to like letters. This is shown in Figure 3.6, for which three panels show the results of simulations of word length $m = 2$ de Bruijn processes with transition probabilities $\{0.9, 0.1, 0.9, 0.1\}$ , $\{0.5, 0.5, 0.5, 0.5\}$ and $\{0.1, 0.9, 0.1, 0.9\}$ respectively. If we compare the top and middle sequences, it is clear how much difference the anti-sticky probabilities can make, as compared to the independent Bernoulli version. The top sequence is seen to be much more structured, with the 0 and 1 values swapping constantly to create a regular anti-clustering pattern. This is in contrast to the bottom sequence where the 0's and 1's are far more sticky to the same letter. These three sequences show that by using the de Bruijn structure we can form a vast variety of different patterns.

So far, I have only discussed examples for which the marginal probabilities for the letters are equal ($\pi(\{0\}) = \pi(\{1\}) = 0.5$); i.e., for the example where $p_{11}^{11} = 0.9$, we must also have $p_{00}^{00} = 0.9$. If the marginals were changed to be $\pi(\{1\}) = 0.8$ and $\pi(\{0\}) = 0.2$, then if $p_{11}^{11} = 0.9$, we must also have $p_{00}^{00} = 0.225$. This is calculated using either the definition, $\pi(\{0\})p_{11}^{11} = (1-\pi(\{0\}))p_{00}^{00}$ or $(1-\pi(\{1\}))p_{11}^{11} = \pi(\{1\})p_{00}^{00}$. Hence, if you know the marginals for 0 and 1, then you can reduce the number of transition probabilities by half. An example using these values is shown in Figure 3.7, with word length $m = 2$. From top to bottom the transition probabilities, $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\}$, are: $\{0.1, 0.8, 0.2, 0.9\}$ and $\{0.775, 0.8, 0.8, 0.9\}$ so that we can

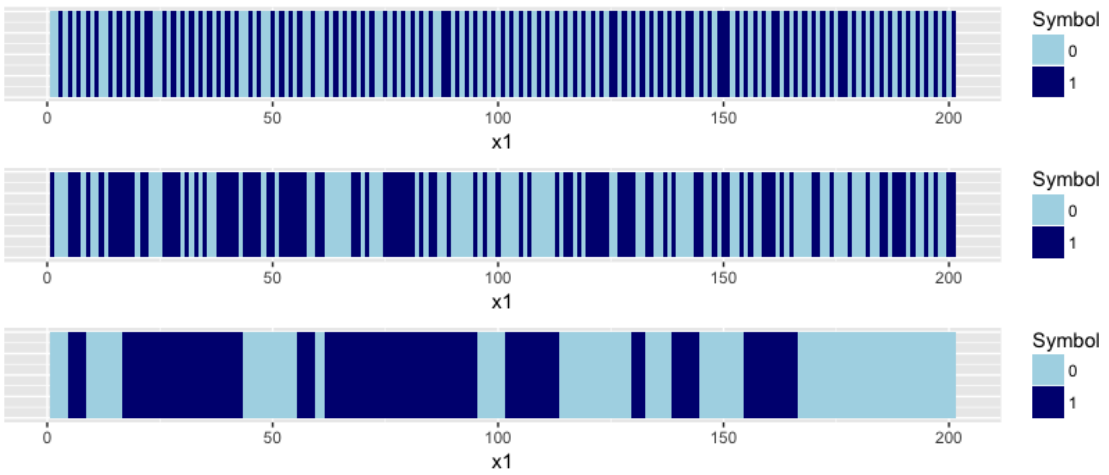



Fig. 3.6 Three samples from de Bruijn processes with letters 0 and 1 to show the differences between an anti-sticky de Bruijn, independent Bernoulli de Bruijn and sticky de Bruijn processes. From top to bottom, the transition probabilities, $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\}$, are: $\{0.9, 0.1, 0.9, 0.1\}$ , $\{0.5, 0.5, 0.5, 0.5\}$ and $\{0.1, 0.9, 0.1, 0.9\}$.

compare a symmetrical example against a non-symmetrical example. Here we can see that there are far more chains of 1's than 0's in the bottom plot (non-symmetrical), but roughly equal numbers of chains of both 1's and 0's in the top plot (symmetrical). The transition probabilities in this example are defined such that the stickiness to 1's is the same, but the number of 1's in the non-symmetric example should be 80% of the total number of letters. This is emphasized by the average run lengths for the two chains being very close in value at 9.96 and 9.68 respectively.

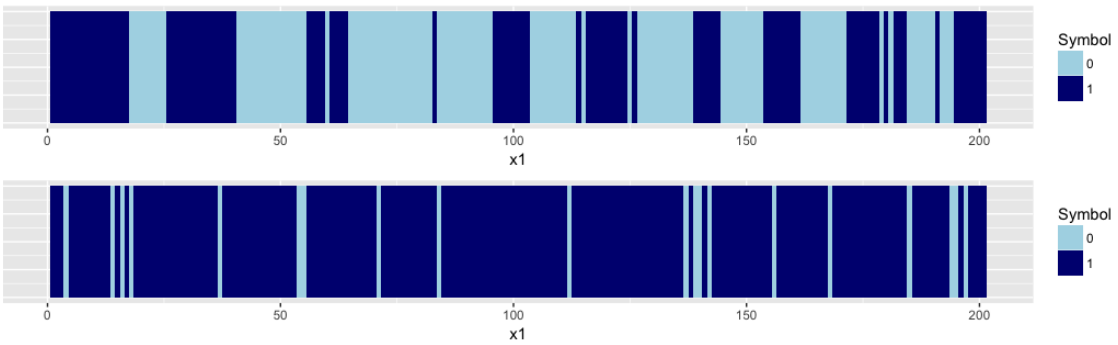



Fig. 3.7 Two samples from de Bruijn processes with letters 0 and 1 to show the differences between symmetric and non-symmetric de Bruijn processes. From top to bottom, the marginal probabilities are: $\pi(\{0\}) = \pi(\{1\}) = 0.5$ and $\pi(\{0\}) = 0.2$, $\pi(\{1\}) = 0.8$, and the transition probabilities, $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\}$, are: $\{0.1, 0.8, 0.2, 0.9\}$ and $\{0.775, 0.8, 0.8, 0.9\}$.

Figure 3.8 shows the effects of the run lengths and letter stickiness when changing the word length, $m$, of the de Bruijn process. From top to bottom, the word lengths used are $m = 1$, $m = 2$, $m = 3$ and $m = 4$. It is much trickier to

make direct comparisons here due to the extra transition probabilities that are introduced at each increase in word length (this plot is just to give a rough idea of the difference a longer word length can make). The transition probabilities for each case are chosen to be fairly similar to each other, but we must keep in mind that we can induce a larger impact on run length when changing both the word length and transition probabilities simultaneously. The transition probabilities used in Figure 3.8 are as follows: $\{p_0^1, p_1^1\} = \{0.2, 0.8\}$, $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\} = \{0.1, 0.8, 0.2, 0.9\}$, $\{p_{000}^{001}, p_{001}^{011}, p_{010}^{101}, p_{011}^{111}, p_{100}^{001}, p_{101}^{011}, p_{110}^{101}, p_{111}^{111}\} = \{0.1, 0.7, 0.4, 0.8, 0.2, 0.6, 0.3, 0.9\}$ and $\{p_{0000}^{0001}, p_{0001}^{0011}, p_{0010}^{0101}, p_{0011}^{0111}, p_{0100}^{1001}, p_{0101}^{1011}, p_{0110}^{1101}, p_{0111}^{1111}, p_{1000}^{0001}, p_{1001}^{0011}, p_{1010}^{0101}, p_{1011}^{0111}, p_{1100}^{1001}, p_{1101}^{1011}, p_{1110}^{1101}, p_{1111}^{1111}\} = \{0.1, 0.6, 0.5, 0.7, 0.3, 0.5, 0.4, 0.8, 0.2, 0.6, 0.5, 0.7, 0.3, 0.5, 0.4, 0.9\}$. As we can see, the length of the word does make a difference to the stickiness of the letter (since there is a slight increase in run length as we move down the different plots), but one of the main advantages of selecting a longer word is the increased structure that can be incorporated through the more intricate transition probabilities.

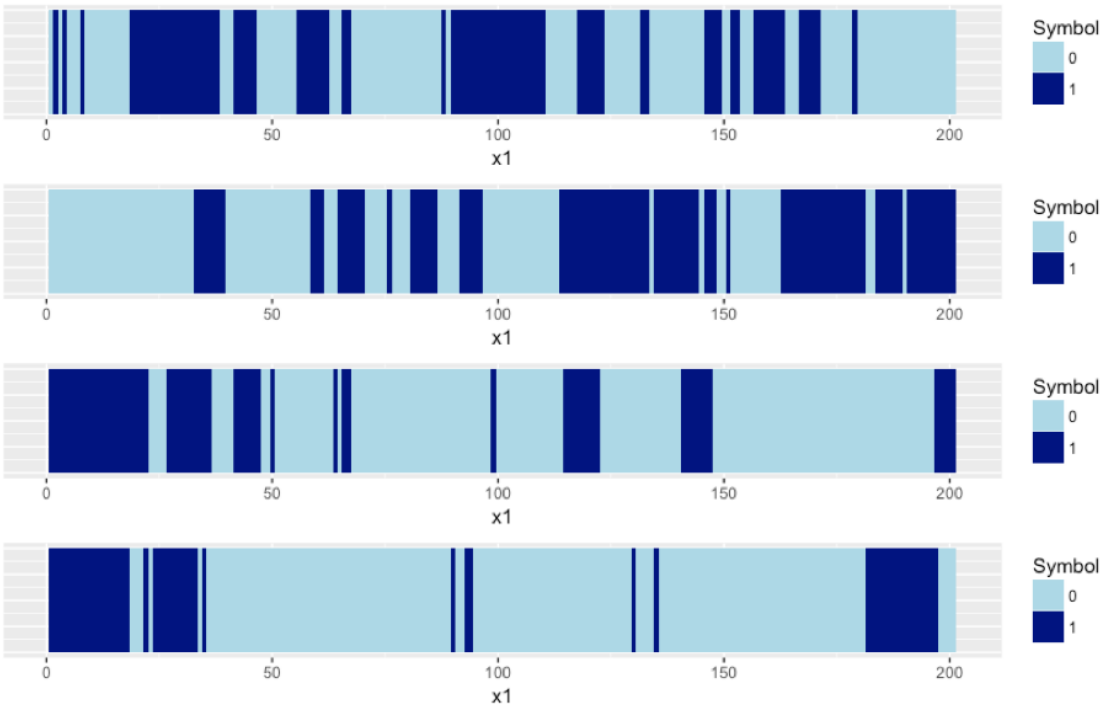



Fig. 3.8 Four samples from de Bruijn processes with letters 0 and 1 to show the effects of changing the the word length, $m$. From top to bottom, the word lengths are $m = 1$, $m = 2$, $m = 3$ and $m = 4$. Transition probabilities are kept fairly equivalent for all examples to show the effects of the word lengths alone.

Normally, when increasing the word length, we can also increase the inherent complexity, and we can add in further structure to the de Bruijn process. However,

we may find that two different de Bruijn processes with different word lengths may actually be equivalent, such that the same sequences could be generated from either process. Two sequences generated from both processes would have the same properties and the run length distributions from both processes would be equivalent. The intermediate transition properties of the larger de Bruijn process are not adding any more structure to the process, so we can actually remove these and convert the larger length word process to that of the shorter word length. A simple example of this is shown in Figure 3.9 where there are four different sequences created with de Bruijn processes with word lengths, $m = 1$, $m = 2$, $m = 3$ and $m = 4$. The transition probabilities are defined such that there is a 10% chance of changing to a different letter and a 90% chance of remaining at the same letter for each of the four different sequences. Here, the intermediate transition probabilities in the larger word length processes are not adding any further structure to the process since there is still a 90% chance of observing the same letter regardless of the word length. This is clearly seen in Figure 3.9 where the run lengths in each of the sequences are all similar to the run lengths generated from the $m = 1$ de Bruijn process (classical Markov).

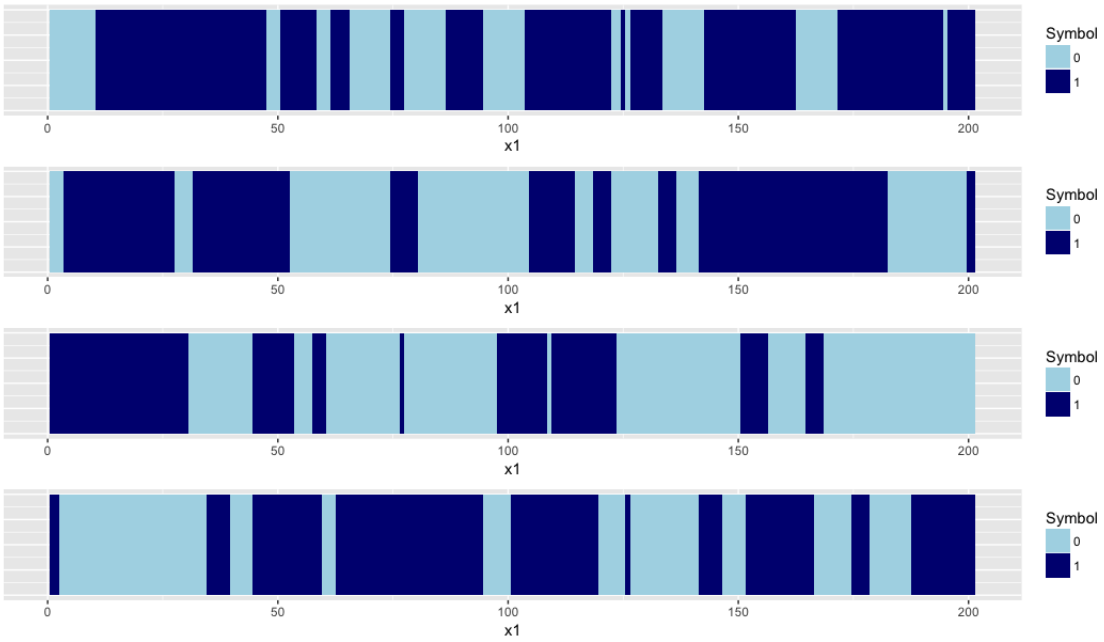



Fig. 3.9 Four samples from de Bruijn processes with letters 0 and 1 to show how certain chains with larger word lengths can be equivalent to chains with shorter word lengths. From top to bottom, the word lengths are $m = 1$, $m = 2$, $m = 3$ and $m = 4$. Transition probabilities are set at 0.1 for adding a different letter (e.g. $p_{110}^{101}$) and 0.9 for adding the same letter (e.g. $p_{011}^{111}$).

### 3.3.2 Markov Properties

In Section 3.2, I showed that the eigenvalues and eigenvectors of the transition matrix for the de Bruijn word Markov chain can offer valuable information. The first eigenvalue is $\lambda = 1$, and the second eigenvalue tells us how fast the Markov chain converges to its stationary distribution (convergence rate). If the transition matrix has the form:

$$T = \begin{pmatrix} q & p & 0 & 0 \\ 0 & 0 & p & q \\ q & p & 0 & 0 \\ 0 & 0 & p & q \end{pmatrix},$$

then the rate of convergence is $q - p$. So, for the example presented in Figure 3.4 where the transition probabilities are $\{0.1, 0.9, 0.1, 0.9\}$, the rate of convergence is $q-p = 0.9-0.1 = 0.8$, indicating that the Markov chain will converge at a reasonably fast rate (around 480 steps to convergence within 0.001 error). For the other examples in Figure 3.4, where the transition probabilities are $\{0.25, 0.75, 0.25, 0.75\}$ and $\{0.05, 0.95, 0.05, 0.95\}$, the convergence rates are 0.5 and 0.9 respectively (with associated steps to convergence of 600 and 330 each within 0.001 error). This shows that the transition probabilities have an influence on the convergence rate; when the letters tend to be more sticky, the Markov chain on the word converges at a higher rate. This is to be expected, since the closer the Markov chain is to being equivalent to independent Bernoulli trials, the more random the sequence is, and convergence to the stationary distribution is slower.

Either powering up the transition matrix, or finding the eigenvector corresponding to the eigenvalue $\lambda = 1$ gives the marginal distribution. For the examples in Figure 3.4 the marginal distributions for the words, $\{\pi(00), \pi(01), \pi(10), \pi(11)\}$, are $\{0.25, 0.25, 0.25, 0.25\}$, $\{0.375, 0.125, 0.125, 0.375\}$, $\{0.45, 0.05, 0.05, 0.45\}$ and $\{0.475, 0.025, 0.025, 0.475\}$ respectively. We can see that all the marginals sum to one so that $\pi(00) + \pi(01) + \pi(10) + \pi(11) = 1$ as expected, and that the individual marginals for 0 and 1 both sum to 0.5 ($\pi(\{0\}) = \pi(\{1\}) = 0.5$) so that $\pi(\{0\}) = \pi(00) + \pi(10) = 0.5$ and $\pi(\{1\}) = \pi(01) + \pi(11) = 0.5$.

The mean recurrence time for a state, $i$, for each of the examples in Figure 3.4 is given by $\frac{1}{\pi_i}$. So, if we are interested in finding the mean recurrence time for the state 11, we just need to find $\frac{1}{\pi(11)}$ for each of the examples in Figure 3.4. This gives times of 4, 2.667, 2.222 and 2.105 respectively. This is again as expected; if the de Bruijn process is very sticky towards 1's then it is much more likely to return to the state 11 again in a short period of time. The mean recurrence times of the state 01 are found to be 4, 8, 20 and 40 for each example respectively. We would expect the de Bruijn processes that are sticky towards 1's to not return to the state 01 very quickly. They are likely to get stuck in the state 11 for a long time, then switch to 10, then stay stuck in the state 00 for a while (if the marginals between the letters 0 and 1 are equal) before returning to the state 01. If instead the de Bruijn process is similar to independent Bernoulli trials, then we would expect the state 01 to appear far more often. For the independent Bernoulli case when the marginals are $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\} = \{0.25, 0.25, 0.25, 0.25\}$, the mean recurrence time is 4 for all states. This is because we are equally likely to go to any state, so on average we would visit all other states before returning to the same one. If we looked at the mean recurrence time for a de Bruijn process with a larger word length, we would find the same patterns, but the overall times would likely be shorter since there are more available states to travel to because the graph is larger.

### 3.3.3 De Bruijn Graphs as Trees

In this section, I will define connections between de Bruijn graphs for a correlated Bernoulli process and a specific type of graphical model called trees (Lauritzen, 1996). Consider a graph $G = (W, E)$, where $W$ is a set of vertices (or nodes) and $E$ is a set of edges such that $E$ is a subset of the set $W \times W$ of ordered pairs of distinct vertices. The edges can either be directed or non-directed, but multiple edges or loops do not exist. Graphs can be extremely useful for visualising certain data or models. By expressing edges between nodes of the graph, we can clearly see any relationships or dependencies in the data.

A tree is a connected, undirected graph without cycles, where a rooted tree is a tree with direction. A single vertex is chosen as the root and all edges are directed away from this point. We can immediately see the connection to de Bruijn graphs since there is a clear direction away from the initial letter, and when we take a random walk on the de Bruijn graph, this is equivalent to taking a specific route down a tree. A de Bruijn graph with letters 0 and 1 expressed as a tree is shown in Figure 3.10.

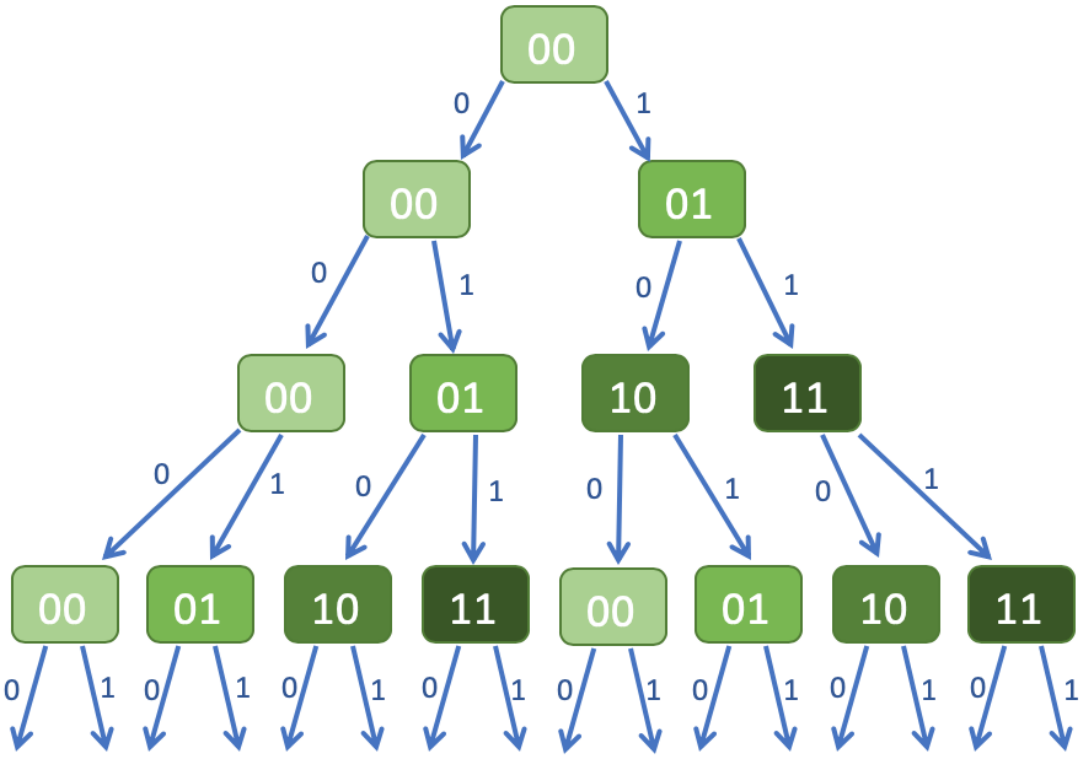



Fig. 3.10 De Bruijn graph with letters 0 and 1 expressed as a section of a tree starting from the root word, 00.

The plot consists of a small section from the tree where the nodes are the de Bruijn words and each edge corresponds to either adding a 0 or a 1 to the sequence. The starting node (the root) in this example is the word 00, but this would be equivalent to any starting word since the tree creates a factorial pattern. We can link trees closer to de Bruijn graphs by attaching the appropriate transition probabilities to each of the edges. In the literature, for example (Sedgewick and Flajolet, 2013), there are different types of trees. In fact, the tree shown in Figure 3.10 is a binary

tree. Binary trees are trees where the nodes have no more than two edges directed out.

De Bruijn graphs as trees is just one example of another way of viewing the graphs themselves. I have not gone into great detail on the subject as it is a topic for future work. By connecting de Bruijn processes to other existing graphical models, we may be able to expand or simplify some of the concepts developed. Therefore I believe research into other existing graph theory is going to be a vital step for the future.

## 3.4 Discussion

In this Chapter, I have given an initial overview of how we can build a correlated Bernoulli process using de Bruijn graphs. There has been very little previous research done in this area, with nothing having the desired structures and simplicity required. The eventual goal is to be able to use de Bruijn processes in conjunction with a classification method such as logistic regression. When taking draws or samples to make classification predictions in logistic regression, we end up neglecting any between input correlation, so it is vital that we can find an alternative to improve on classification predictions.

De Bruijn graphs are a way of creating sequences of 0's and 1's such that we can control the spread of correlation across neighbouring points. They have connections to Markov chains, whose properties can help to define quantities such as the stationary distribution, mean recurrence time and letter run lengths. In this chapter I have defined the de Bruijn process and shown examples including samples of the types of sequences that we can generate from them. I have also given examples on how we can change the structures of the sequences so that we can force the 0's and 1's to be more sticky and clump together for like letter. I feel that there are many areas for further research in this area which will be discussed in detail in Chapter 7.

In the following chapter I will be developing the de Bruijn process further to look at the run length distributions of letters, as well as inference. In Chapter 5, I will show possible methods to expand de Bruijn graphs to higher dimensions and in

Chapter 6 I will present thoughts on moving towards a non-directional de Bruijn graph. My aim is to fully remove the direction out of de Bruijn graphs since it is not always natural to have a direction when considering the chains (or grids in Chapter 5) of 0's and 1's.

# Chapter 4

# De Bruijn Process Properties and Inference

## 4.1 Introduction

In Chapter 3, I outlined an approach to producing a correlated Bernoulli process using de Bruijn graphs. I demonstrated how to create chains of 0's and 1's, for which we can introduce structure so that the sequence becomes far more 'sticky' to the same letters. Similarly, we can reverse this so that we can produce the inverse effect: creating an 'anti-sticky' sequence that constantly swaps between letters. We will first be discussing the run length distribution of the de Bruijn process. The run length distribution specifies the chance of observing each length of runs of 1's (or 0's) in a sequence for a given de Bruijn process. The number of consecutive 1's (or 0's) in a row will hence give a measure of how sticky a sequence generated from a specific de Bruijn process is likely to be. The structure of the run length problem allows me to use well established results developed for the geometric distribution to help derive results. Examples are given comparing the theoretical expected run lengths against the simulated run lengths obtained in Section 3.2.1 for Figures 3.4, 3.6, 3.7 and 3.9.

Secondly, I have developed a method of inference which is discussed in Section 4.3. Given a sequence of 0's and 1's, we want to be able to estimate the de Bruijn process that was used to create it. This involves estimating both the word length, $m$, and the transition probabilities, $p$, that determine the correlated structure of the

corresponding Markov chains. Both frequentist and Bayesian solutions are given, along with an example to illustrate the method.

## 4.2 Run Length

A property of interest within the de Bruijn process is the run lengths of particular letters, and in particular whether these run lengths change with different word lengths and/or transition probabilities. For the remainder of this section, I will discuss, without loss if generality, the run lengths of 1's for my de Bruijn processes discussed in Section 3.2.1. The aim is to improve on current run length literature such as Fu and Koutras (1994) who focus on the run lengths in sequences of independent Bernoulli trials. A run is defined as the number of consecutive 1's in a row bounded by a 0 at each end. We saw in Chapter 3 that by changing both the length of the de Bruijn word and the transition probabilities, we are able to create both sticky and anti-sticky sequences of 0's and 1's. We also know that the very sticky de Bruijn processes produce far longer run lengths of 1's than those that are closer to independent Bernoulli trials. Therefore, by analysing the run lengths of different de Bruijn processes we will be able to give some quantification of the stickiness of different sequences, and identify the de Bruijn graphs that created them. As well as the distribution itself, properties of the run length will be discussed including the expectation, variance and generating functions. As a reminder, $\pi(i)$ represents the marginal probability of the word $i$ (or $\pi(\{i\})$ for the letter $i$) and $p_i^j$ represents the probability of transitioning from the word $i$ to the word $j$.

### 4.2.1 Run Length Distribution

The run length distribution for a word length $m = 2$ de Bruijn process with transition probabilities $\{p_{01}^{10}, p_{01}^{11}, p_{11}^{10}, p_{11}^{11}\}$ is given in Lemma 4.1. This distribution gives the probability of a run of 1's of length $n$, for any $n \in \mathbb{N}^+$. The probability of a single 1 is simply the transition probability $p_{01}^{10}$, since this gives us the probability of transitioning from the word 01 to the word 10, giving the sequence 010. For higher run lengths we again start with the word 01, but now transition to the word 11, with

probability $p_{01}^{11}$. For a run of length two, we would then finish with the probability $p_{11}^{10}$, but for a run length of three, we would further transition to the same word 11 with transition probability $p_{11}^{11}$. For run lengths of greater than three, we would continue to return to the word 11 until a run length of $n$ is reached. Hence, in Lemma 4.1 this transition probability is raised to the power $n-2$. The power is taken to be $n-2$ here instead of $n$ since a run length of two is already created with the other two transition probabilities in the equation, and so two must be removed to compensate. Every run must finish with the probability $p_{11}^{10}$ since the zero must be there to finish the run off. As said previously in Chapter 3, if the de Bruijn process is very sticky to 1's, then the value of $p_{11}^{11}$ will be close to one and the probability of long run lengths will be high. Also, for small values of $p_{11}^{11}$, the probability of long run lengths will be small. Note that this distribution is conditional on a run existing. For future work this could be extended to also include run lengths of length zero.

**Lemma 4.1** (Run Length Distribution, $m = 2$)**.**

$$P(\textbf{run length} = n) = \begin{cases} p_{01}^{10} & \textit{for } n = 1 \\ p_{01}^{11}(p_{11}^{11})^{n-2}p_{11}^{10} & \textit{for } n \geq 2, \end{cases}$$

*Proof.* $P(\textbf{run length} = n) = p_{01}^{10}$ for $n = 1$. For $n \geq 2$, the run starts with the word 01 and transitions to 11 with probability $p_{01}^{11}$ creating the sequence 011. The run continues by transitioning to the word, 11, with probability $p_{11}^{11}$, $n-2$ times to create a run of length $n$. The run ends with the first zero with probability $p_{11}^{10}$, hence: $P(\textbf{run length} = n) = p_{01}^{11}(p_{11}^{11})^{n-2}p_{11}^{10}$, for $n \geq 2$.

For the above equation to be the appropriate distribution function, for all values of $n$ from 1 to $\infty$, then it must be true that $\sum_{n=1}^{\infty} P(\textbf{run length} = n) = 1$:

$$\begin{aligned}\sum_{n=1}^{\infty} P(\textbf{run length} = n) &= p_{01}^{10} + \sum_{n=2}^{\infty} p_{01}^{11}(p_{11}^{11})^{n-2}p_{11}^{10} \\ &= p_{01}^{10} + p_{01}^{11}p_{11}^{10}\sum_{n=0}^{\infty}(p_{11}^{11})^{n} \\ &= p_{01}^{10} + \frac{p_{01}^{11}p_{11}^{10}}{1-p_{11}^{11}} \\ &= p_{01}^{10} + \frac{p_{01}^{11}p_{11}^{10}}{p_{11}^{10}} \\ &= p_{01}^{10} + p_{01}^{11} \\ &= 1\end{aligned}$$

since, $\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$ when $|x| < 1$, $p_{11}^{10} + p_{11}^{11} = 1$ and $p_{01}^{10} + p_{01}^{11} = 1$. □

In Theorem 4.2 the run length distribution is extended for de Bruijn words of length three and higher using the decimal representation of the binary notation discussed in Section 3.2.1. Although this appears to be far more complicated than the $m = 2$ version, it follows the same pattern. We retain the term that represents when the de Bruijn process continues to return to the state that only consists of 1's $\left(p_{2^m-1}^{2^m-1}\right)$, but there is now a longer run in period until this point. To start a sequence of 1's off, the first word must take the form $*01$; the run starts with the first 1 after a 0, and it makes no difference which letters come before. Hence, $*$ can represent any possible sequence of length $m-2$, and therefore there are $2^{m-2}$ possible starting words that will eventually lead into the same run of 1's. Since there are $2^{m-2}$ starting positions, we must average over all possibilities using the law of total probability (Feller, 1950) to get the full run length distribution. The equation for total probability is used as follows:

$$P(\textbf{run length} = n) = \sum_{i=0}^{2^{m-2}-1} P(\textbf{run length} = n|*_i)\pi(*_i), \tag{4.1}$$

where $\pi(*_i)$ is the marginal probability for the $i$th starting sequence. Since the sequences represented by $*$ are of length $m-2$, the marginal probabilities for these sequences must be found given that we are working with a de Bruijn process with

word length $m$. We do this by applying the law of total probability again, such that:

$$\pi(*_i) = \sum_{j=0}^{2^{m-2}-1} P(*_i|j)\pi(j),$$

making it possible to find the probabilities of the sequences $*_i$ by averaging over all of the possible starting words that could produce this sequence.

For example, assume we have a de Bruijn process of word length $m = 3$. To begin our run of 1's, we can either start with the word 001 or 101. Therefore, using Equation (4.1), we need to find $\pi(\{0\})$ and $\pi(\{1\})$ where,

$$\begin{aligned}\pi(\{0\}) = p_{000}^{000}\pi(000) + p_{001}^{010}\pi(001) + p_{010}^{100}\pi(010) + p_{011}^{110}\pi(011) \\ + p_{100}^{000}\pi(100) + p_{101}^{010}\pi(101) + p_{110}^{100}\pi(110) + p_{111}^{110}\pi(111).\end{aligned} \tag{4.2}$$

As a reminder, $\pi(\{i\})$ represents the marginal probability of $i$ if $i$ is a letter, whilst $\pi(i)$ represents the marginal probability of $i$ if $i$ as a word.

For larger word lengths, there are not only more possible starting positions, but the run-in or burn-in period is also longer. This is due to the fact that it takes longer for a larger word to reach a sequence of all 1's. Thus, in Theorem 4.2, I have chosen to write the cases where $n$ is smaller than the word length, $m$, separately as there is a slightly different pattern. In these cases, the run does not reach the state with all 1's before it hits a 0, and instead transitions to a word of the form $*10$. All runs of length $n$ which are greater than the word length $m$ end up reaching the all 1 state, and then finish the run with the first 0, resulting in a word of the form 11..10 with transition probability $p_{2^m-1}^{2^m-2}$. The state with all 1's must also be raised to the power $n - m$ since a length of $m$ 1's is obtained in the run up period.

**Theorem 4.2** (Run Length Distribution, $m \geq 3$)**.**

$$
P(\textbf{run length} = n)
$$

$$
= \begin{cases}
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} & \text{for } n = 1 \\
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} & \text{for } n = 2 : m-1 \\
\qquad \times \left[\prod_{j=1}^{n-1} p_{2^{j+2}(i \bmod 2^{m-j-3})+(2^{j+2}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=n-1}}\right] & \\
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} & \\
\qquad \times \left[\prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-3})+(2^{j+2}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)}\right] & \text{for } n \geq m. \\
\qquad \times \left[\left(p_{2^m-1}^{2^m-1}\right)^{n-m} p_{2^m-1}^{2^m-2}\right] &
\end{cases}
$$

where,

$$
\pi(i) = \sum_{j=0}^{2^m-1} \prod_{k=0}^{m-3} \left[ p_{2^k(j \bmod 2^{m-k})+\sum_{s=1}^{k} 2^{k-s}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]}^{2^{k+1}(j \bmod 2^{m-k-1})+\sum_{s=1}^{k+1} 2^{k-s+1}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]} \right] \pi(j)
$$

*Proof.* See Appendix A □

For example, consider again the run length distribution for a length $m = 3$ de Bruijn process. We can start the run off with either the word 001 or 101 so that the run length distribution becomes:

$$
\begin{aligned}
P(\textbf{run length} = n) = P(&\textbf{run length} = n|\{0\})\pi(\{0\}) \\
&+ P(\textbf{run length} = n|\{1\})\pi(\{1\}),
\end{aligned}
$$

where $P(\textbf{run length} = n|\{0\})$ represents the probability of getting a run of length $n$ conditional on starting at the letter 0 with word 001. To find $\pi(\{0\})$ (and similarly $\pi(\{1\})$) we use the definition in Equation (4.2). This converts the length $m-2$ marginal distribution into an expression in terms of the length $m$ marginals. Then, providing the underlying Markov chain is stationary, we can find the length $m$ marginals by powering up the transition matrix or by looking at the eigenvalues and eigenvectors (see Section 3.2.1). Note that it is easier to understand these problems

by looking at the corresponding de Bruijn graph; the graph for $m = 3$ is shown in Figure 3.3.

For a run length of a single 1 (keeping $m = 3$), we consequently have the probability $p_{001}^{010}\pi(\{0\}) + p_{101}^{010}\pi(\{1\})$, which create the sequences 0010 and 1010 respectively. For a run length of two 1's we instead make two transitions such that the next letters to include are a 1 and a 0. Such a run has probability $p_{001}^{011}p_{011}^{110}\pi(\{0\}) + p_{101}^{011}p_{011}^{110}\pi(\{1\})$. For run lengths of three or more, we now have a run up long enough to reach the state 111, and we proceed in a similar way as above, but also include powers of $p_{111}^{111}$ until we reach the desired run length. A run of length $n$ then has probability $p_{001}^{011}p_{011}^{111}\left(p_{111}^{111}\right)^{n-3}\pi(\{0\}) + p_{101}^{011}p_{011}^{111}\left(p_{111}^{111}\right)^{n-3}\pi(\{1\})$. The power we apply to $p_{111}^{111}$ is now $n-3$, as the run up to $p_{111}^{111}$ includes three 1's.

It is clear that the run length distribution has a connection to the discrete geometric distribution (Feller, 1950; Johnson et al., 2005). The geometric distribution gives the probability that the first success (1) from a set of independent trials, $X$, with probability of success, $p$, happens on the $(k+1)$th trial. The density is given by: $P(X = k) = (1-p)^k p$ for $k = 0, 1, 2, ...$, giving the probability distribution of the number of independent Bernoulli trials needed to get the first success.

We can consider the run length distribution as a generalised geometric distribution due to the burn-in period that is required for the de Bruijn process. Although the geometric distribution considers independent trials, whilst I want to retain correlation as a vital part of the run length, we can still observe the transitions in a geometric way. Since the transitions have a specific ordering to make the words fit into the de Bruijn structure, there is still a correlation enforced in this way. I will be using this link to the geometric distribution to help in following calculations in this chapter.

### 4.2.2 Expected Run Length

From the run length distribution, we can now calculate subsequent measures. I first consider the expectation (Feller, 1950), where the expected run length for a length $m = 2$ de Bruijn process is given in Lemma 4.3. Due to the connections between the de Bruijn process run length and the geometric distribution, we can use results

for the geometric distribution to make simplifications to expressions. The geometric distribution function takes the form, $P(X = x) = (1-p)^x p$, for $x = 0, 1, 2, 3, ...$, meaning that we can use the expectation, $E[X] = \sum_x xP(X = x) = \sum_x x(1-p)^x p = \frac{q}{p}$, to help make simplifications. Even though the smallest run is of length one, I use the geometric distribution starting from $x = 0$ since raising $p_{11}^{11}$ to the power zero still gives a positive run length.

**Lemma 4.3** (Expected Run Length, $m = 2$)**.**

$$E\left[\textbf{run length}\right] = p_{01}^{10} + \frac{p_{01}^{11}\left(1-(p_{11}^{10})^2\right)}{p_{11}^{11}p_{11}^{10}}.$$

*Proof.*

$$\begin{aligned}
E\left[\textbf{run length}\right] &= \sum_{n=1}^{\infty} n \times P(\textbf{run length} = n) \\
&= p_{01}^{10} + \sum_{n=2}^{\infty} n p_{01}^{11}(p_{11}^{11})^{n-2} p_{11}^{10} \\
&= p_{01}^{10} + \frac{p_{01}^{11}}{(p_{11}^{11})^2} \sum_{n=2}^{\infty} n(p_{11}^{11})^n p_{11}^{10} \\
&= p_{01}^{10} + \frac{p_{01}^{11}}{(p_{11}^{11})^2} \left[\sum_{n=0}^{\infty} n(1-p_{11}^{10})^n p_{11}^{10} - p_{11}^{11}p_{11}^{10}\right] \\
&= p_{01}^{10} + \frac{p_{01}^{11}}{(p_{11}^{11})^2} \left[\frac{p_{11}^{11}}{p_{11}^{10}} - p_{11}^{11}p_{11}^{10}\right] \\
&= p_{01}^{10} + \frac{p_{01}^{11}\left(1-(p_{11}^{10})^2\right)}{p_{11}^{11}p_{11}^{10}}.
\end{aligned}$$

since $E[X] = \sum_x xP(X = x) = \sum_x x(1-p)^x p = \frac{1-p}{p}$ for a geometric distribution with $P(X = x) = (1-p)^x p$ and $x = 0, 1, 2, 3, ....$ □

Following from this, the expressions for the expected run lengths for de Bruijn processes of length $m = 3$ and $m = 4$ are given in Lemmas 4.4 and 4.5 respectively. I have included these so that the reader can get a better idea of how the de Bruijn run length distribution works.

**Lemma 4.4** (Expected Run Length, $m = 3$)**.**

$$
\begin{aligned}
&E\,[\textbf{run length}] \\
&= \pi(\{0\})p_{001}^{010} + \pi(\{1\})p_{101}^{010} + 2\left[\pi(\{0\})p_{001}^{011}p_{011}^{110} + \pi(\{1\})p_{101}^{011}p_{011}^{111}\right] \\
&\quad + \frac{1}{(p_{111}^{111})^3}\left[\pi(\{0\})p_{001}^{011}p_{011}^{111} + \pi(\{1\})p_{101}^{011}p_{011}^{111}\right]\left[\frac{p_{111}^{111}}{p_{111}^{110}} - p_{111}^{111}p_{111}^{110} - 2(p_{111}^{111})^2 p_{111}^{110}\right],
\end{aligned}
$$

where,

$$
\begin{aligned}
\pi(\{0\}) &= p_{000}^{000}\pi(000) + p_{001}^{010}\pi(001) + p_{010}^{100}\pi(010) + p_{011}^{110}\pi(011) \\
&\quad + p_{100}^{000}\pi(100) + p_{101}^{010}\pi(101) + p_{110}^{100}\pi(110) + p_{111}^{110}\pi(111) \\
\pi(\{1\}) &= p_{000}\pi(001) + p_{001}^{011}\pi(001) + p_{010}^{101}\pi(010) + p_{011}^{111}\pi(011) \\
&\quad + p_{100}^{001}\pi(100) + p_{101}^{011}\pi(101) + p_{110}^{101}\pi(110) + p_{111}^{111}\pi(111).
\end{aligned}
$$

*Proof.*

$$
\begin{aligned}
&E\left[\textbf{run length}\right] = \sum_{n=1}^{\infty} n \times P(\textbf{run length} = n) \\
&= \pi(\{0\})p_{001}^{010} + \pi(\{1\})p_{101}^{010} + 2\left[\pi(\{0\})p_{001}^{011}p_{011}^{110} + \pi(\{1\})p_{101}^{011}p_{011}^{110}\right] \\
&\quad + \left[\pi(\{0\})p_{001}^{011}p_{011}^{111} + \pi(\{1\})p_{101}^{011}p_{011}^{111}\right] \sum_{n=3}^{\infty} n(p_{111}^{111})^{n-3}p_{111}^{110} \\
&= \pi(\{0\})p_{001}^{010} + \pi(\{1\})p_{101}^{010} + 2\left[\pi(\{0\})p_{001}^{011}p_{011}^{110} + \pi(\{1\})p_{101}^{011}p_{011}^{110}\right] \\
&\quad + \frac{1}{(p_{111}^{111})^3}\left[\pi(\{0\})p_{001}^{011}p_{011}^{111} + \pi(\{1\})p_{101}^{011}p_{011}^{111}\right]\left[\sum_{n=0}^{\infty} n(p_{111}^{111})^n p_{111}^{110} - p_{111}^{111}p_{111}^{110}\right. \\
&\quad \left. - 2(p_{111}^{111})^2 p_{111}^{110}\right] \\
&= \pi(\{0\})p_{001}^{010} + \pi(\{1\})p_{101}^{010} + 2\left[\pi(\{0\})p_{001}^{011}p_{011}^{110} + \pi(\{1\})p_{101}^{011}p_{011}^{111}\right] \\
&\quad + \frac{1}{(p_{111}^{111})^3}\left[\pi(\{0\})p_{001}^{011}p_{011}^{111} + \pi(\{1\})p_{101}^{011}p_{011}^{111}\right]\left[\frac{p_{111}^{111}}{p_{111}^{110}} - p_{111}^{111}p_{111}^{110} - 2(p_{111}^{111})^2 p_{111}^{110}\right],
\end{aligned}
$$

where,

$$
\begin{aligned}
\pi(\{0\}) &= p_{000}^{000}\pi(000) + p_{001}^{010}\pi(001) + p_{010}^{100}\pi(010) + p_{011}^{110}\pi(011) \\
&\quad + p_{100}^{000}\pi(100) + p_{101}^{010}\pi(101) + p_{110}^{100}\pi(110) + p_{111}^{110}\pi(111) \\
\pi(\{1\}) &= p_{000}\pi(001) + p_{001}^{011}\pi(001) + p_{010}^{101}\pi(010) + p_{011}^{111}\pi(011) \\
&\quad + p_{100}^{001}\pi(100) + p_{101}^{011}\pi(101) + p_{110}^{101}\pi(110) + p_{111}^{111}\pi(111).
\end{aligned}
$$

since $E[X] = \sum_x xP(X = x) = \sum_x x(1-p)^x p = \frac{1-p}{p}$ for a geometric distribution with $P(X = x) = (1-p)^x p$ and $x = 0, 1, 2, 3, ....$ □

**Lemma 4.5** (Expected Run Length, $m = 4$)**.**

$$
\begin{aligned}
&E\,[\textbf{run length}]\\
&= \pi(00)p_{0001}^{0010} + \pi(01)p_{0101}^{1010} + \pi(10)p_{1001}^{0010} + \pi(11)p_{1101}^{1010}\\
&\quad + 2\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0110} + \pi(01)p_{0101}^{1011}p_{1011}^{0110} + \pi(10)p_{1001}^{0011}p_{0011}^{0110}\\
&\qquad + \pi(11)p_{1101}^{1011}p_{1011}^{0110}\Big]\\
&\quad + 3\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0111}p_{0111}^{1110} + \pi(01)p_{0101}^{1011}p_{1011}^{0111}p_{0111}^{1110} + \pi(10)p_{1001}^{0011}p_{0011}^{0111}p_{0111}^{1110}\\
&\qquad + \pi(11)p_{1101}^{1011}p_{1011}^{0111}p_{0111}^{1110}\Big]\\
&\quad + \frac{1}{(p_{1111}^{1111})^4}\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0111}p_{0111}^{1111} + \pi(01)p_{0101}^{1011}p_{1011}^{0111}p_{0111}^{1111} + \pi(10)p_{1001}^{0011}p_{0011}^{0111}p_{0111}^{1111}\\
&\qquad + \pi(11)p_{1101}^{1011}p_{1011}^{0111}p_{0111}^{1111}\Big]\\
&\qquad \times \left[\frac{p_{1111}^{1111}}{p_{1111}^{1110}} - p_{1111}^{1111}p_{1111}^{1110} - 2(p_{1111}^{1111})^2p_{1111}^{1110} - 3(p_{1111}^{1111})^3p_{1111}^{1110}\right]
\end{aligned}
$$

where,

$$
\pi(i) = \sum_{j=0}^{15}\left[p_{(j \bmod 16)}^{2(j \bmod 8)+\left[(\frac{1}{2}(i-(i \bmod 2)))\bmod 2\right]} \times p_{2(j \bmod 8)+\left[(\frac{1}{2}(i-(i \bmod 2)))\bmod 2\right]}^{4(j \bmod 4)+2\left[(\frac{1}{2}(i-(i \bmod 2)))\bmod 2\right]+i \bmod 2}\right]\pi(j)
$$

for i $\in \{00, 01, 10, 11\}$ and where the decimal representation of binary notation is used (i.e. i $\in \{0, 1, 2, 3\}$).

*Proof.*

$$
\begin{aligned}
&E\left[\textbf{run length}\right] = \sum_{n=1}^{\infty} n \times P(\textbf{run length} = n) \\
&= \pi(00)p_{0001}^{0010} + \pi(01)p_{0101}^{1010} + \pi(10)p_{1001}^{0010} + \pi(11)p_{1101}^{1010} \\
&\quad + 2\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0110} + \pi(01)p_{0101}^{1011}p_{1011}^{0110} + \pi(10)p_{1001}^{0011}p_{0011}^{0110} \\
&\quad\quad + \pi(11)p_{1101}^{1011}p_{1011}^{0110}\Big] \\
&\quad + 3\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0111}p_{0111}^{1110} + \pi(01)p_{0101}^{1011}p_{1011}^{0111}p_{0111}^{1110} + \pi(10)p_{1001}^{0011}p_{0011}^{0111}p_{0111}^{1110} \\
&\quad\quad + \pi(11)p_{1101}^{1011}p_{1011}^{0111}p_{0111}^{1110}\Big] \\
&\quad + \Big[\pi(00)p_{0001}^{0011}p_{0011}^{0111}p_{0111}^{1111} + \pi(01)p_{0101}^{1011}p_{1011}^{0111}p_{0111}^{1111} + \pi(10)p_{1001}^{0011}p_{0011}^{0111}p_{0111}^{1111} \\
&\quad\quad + \pi(11)p_{1101}^{1011}p_{1011}^{0111}p_{0111}^{1111}\Big] \times \sum_{n=4}^{\infty} n(p_{1111}^{1111})^{n-4}p_{1111}^{1110} \\
&= \pi(00)p_{0001}^{0010} + \pi(01)p_{0101}^{1010} + \pi(10)p_{1001}^{0010} + \pi(11)p_{1101}^{1010} \\
&\quad + 2\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0110} + \pi(01)p_{0101}^{1011}p_{1011}^{0110} + \pi(10)p_{1001}^{0011}p_{0011}^{0110} \\
&\quad\quad + \pi(11)p_{1101}^{1011}p_{1011}^{0110}\Big] \\
&\quad + 3\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0111}p_{0111}^{1110} + \pi(01)p_{0101}^{1011}p_{1011}^{0111}p_{0111}^{1110} + \pi(10)p_{1001}^{0011}p_{0011}^{0111}p_{0111}^{1110} \\
&\quad\quad + \pi(11)p_{1101}^{1011}p_{1011}^{0111}p_{0111}^{1110}\Big] \\
&\quad + \frac{1}{(p_{1111}^{1111})^4}\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0111}p_{0111}^{1111} + \pi(01)p_{0101}^{1011}p_{1011}^{0111}p_{0111}^{1111} + \pi(10)p_{1001}^{0011}p_{0011}^{0111}p_{0111}^{1111} \\
&\quad\quad + \pi(11)p_{1101}^{1011}p_{1011}^{0111}p_{0111}^{1111}\Big] \times \Big[\sum_{n=0}^{\infty} n(p_{1111}^{1111})^n p_{1111}^{1110} - p_{1111}^{1111}p_{1111}^{1110} \\
&\quad\quad - 2(p_{1111}^{1111})^2 p_{1111}^{1110} - 3(p_{1111}^{1111})^3 p_{1111}^{1110}\Big] \\
&= \pi(00)p_{0001}^{0010} + \pi(01)p_{0101}^{1010} + \pi(10)p_{1001}^{0010} + \pi(11)p_{1101}^{1010} \\
&\quad + 2\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0110} + \pi(01)p_{0101}^{1011}p_{1011}^{0110} + \pi(10)p_{1001}^{0011}p_{0011}^{0110} \\
&\quad\quad + \pi(11)p_{1101}^{1011}p_{1011}^{0110}\Big] \\
&\quad + 3\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0111}p_{0111}^{1110} + \pi(01)p_{0101}^{1011}p_{1011}^{0111}p_{0111}^{1110} + \pi(10)p_{1001}^{0011}p_{0011}^{0111}p_{0111}^{1110} \\
&\quad\quad + \pi(11)p_{1101}^{1011}p_{1011}^{0111}p_{0111}^{1110}\Big] \\
&\quad + \frac{1}{(p_{1111}^{1111})^4}\Big[\pi(00)p_{0001}^{0011}p_{0011}^{0111}p_{0111}^{1111} + \pi(01)p_{0101}^{1011}p_{1011}^{0111}p_{0111}^{1111} + \pi(10)p_{1001}^{0011}p_{0011}^{0111}p_{0111}^{1111} \\
&\quad\quad + \pi(11)p_{1101}^{1011}p_{1011}^{0111}p_{0111}^{1111}\Big] \\
&\quad\quad \times \left[\frac{p_{1111}^{1111}}{p_{1111}^{1110}} - p_{1111}^{1111}p_{1111}^{1110} - 2(p_{1111}^{1111})^2 p_{1111}^{1110} - 3(p_{1111}^{1111})^3 p_{1111}^{1110}\right]
\end{aligned}
$$

where,

$$\pi(i) = \sum_{j=0}^{15} \left[ p_{(j \bmod 16)}^{2(j \bmod 8)+\left[(\frac{1}{2}(i-(i \bmod 2)))\bmod 2\right]} \times p_{2(j \bmod 8)+\left[(\frac{1}{2}(i-(i \bmod 2)))\bmod 2\right]}^{4(j \bmod 4)+2\left[(\frac{1}{2}(i-(i \bmod 2)))\bmod 2\right]+i \bmod 2} \right] \pi(j)$$

since $E[X] = \sum_x xP(X = x) = \sum_x x(1-p)^x p = \frac{1-p}{p}$ for a geometric distribution with $P(X = x) = (1-p)^x p$ and $x = 0, 1, 2, 3, ....$ □

Finally, the expected run length for de Bruijn processes with word length three or higher is given in Theorem 4.6.

**Theorem 4.6** (Expected Run Length, $m \geq 3$)**.**

$$\begin{aligned} & E\,[\textbf{run length}] \\ &= \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\ &\quad + \sum_{k=2}^{m-1} k \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{k-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=k-1}} \right] \right] \\ &\quad + \frac{1}{\left(p_{2^m-1}^{2^m-1}\right)^m} \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] \right] \\ &\qquad \times \left[ \frac{p_{2^m-1}^{2^m-1}}{p_{2^m-1}^{2^m-2}} - \sum_{n=1}^{m-1} n \left(p_{2^m-1}^{2^m-1}\right)^n p_{2^m-1}^{2^m-2} \right] \end{aligned}$$

where,

$$\pi(i) = \sum_{j=0}^{2^m-1} \prod_{k=0}^{m-3} \left[ p_{2^k(j \bmod 2^{m-k})+\sum_{s=1}^{k} 2^{k-s}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))mod\ 2]}^{2^{k+1}(j \bmod 2^{m-k-1})+\sum_{s=1}^{k+1} 2^{k-s+1}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))mod\ 2]} \right] \pi(j)$$

*Proof.*

$$
\begin{aligned}
&E\left[\textbf{run length}\right] = \sum_{n=1}^{\infty} n \times P(\textbf{run length} = n) \\
&= \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
&\quad + \sum_{k=2}^{m-1} k \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{k-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=k-1}} \right] \right] \\
&\quad + \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] \right] \\
&\qquad \times \left[ \sum_{n=m}^{\infty} n \left(p_{2^m-1}^{2^m-1}\right)^{n-m} p_{2^m-1}^{2^m-2} \right] \\
&= \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
&\quad + \sum_{k=2}^{m-1} k \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{k-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=k-1}} \right] \right] \\
&\quad + \frac{1}{\left(p_{2^m-1}^{2^m-1}\right)^m} \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] \right] \\
&\qquad \times \left[ \sum_{n=0}^{\infty} n \left(p_{2^m-1}^{2^m-1}\right)^n p_{2^m-1}^{2^m-2} - \sum_{j=1}^{m-1} j \left(p_{2^m-1}^{2^m-1}\right)^j p_{2^m-1}^{2^m-2} \right] \\
&= \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
&\quad + \sum_{k=2}^{m-1} k \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{k-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=k-1}} \right] \right] \\
&\quad + \frac{1}{\left(p_{2^m-1}^{2^m-1}\right)^m} \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] \right] \\
&\qquad \times \left[ \frac{p_{2^m-1}^{2^m-1}}{p_{2^m-1}^{2^m-2}} - \sum_{j=1}^{m-1} j \left(p_{2^m-1}^{2^m-1}\right)^j p_{2^m-1}^{2^m-2} \right]
\end{aligned}
$$

where,

$$
\pi(i) = \sum_{j=0}^{2^m-1} \prod_{k=0}^{m-3} \left[ p_{2^k(j \bmod 2^{m-k})+\sum_{s=1}^{k} 2^{k-s}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]}^{2^{k+1}(j \bmod 2^{m-k-1})+\sum_{s=1}^{k+1} 2^{k-s+1}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]} \right] \pi(j)
$$

since $E[X] = \sum_x xP(X = x) = \sum_x x(1-p)^x p = \frac{1-p}{p}$ for a geometric distribution with $P(X = x) = (1-p)^x p$ and $x = 0, 1, 2, 3, ....$ □

### 4.2.3 Variance

The next important measure to consider is the variance of the run length distribution. We can still use results from the geometric distribution to help simplify calculations, where, for the distribution function, $P(X = x) = (1-p)^x p$ for $x = 0, 1, 2, ...$, it is known that, $E[X^2] = \sum_x x^2 P(X = x) = \sum_x x^2 (1-p)^x p = \frac{(1-p)(2-p)}{p^2}$. Then, by using the results of Section 4.2.2, and the well known result, Var[**run length**] = $E[\textbf{run length}^2] - (E[\textbf{run length}])^2$, we can calculate the variance for the run length distribution. The squared expectation of the run length distribution for the $m = 2$ word length case is given in Lemma 4.7.

**Lemma 4.7** (Squared Expectation of Run Length, $m = 2$)**.**

$$E[\textbf{run length}^2] = p_{01}^{10} + \frac{p_{01}^{11}\left(2 - p_{11}^{10} - \left(p_{11}^{10}\right)^3\right)}{p_{11}^{11}\left(p_{11}^{10}\right)^2}$$

*Proof.*

$$\begin{aligned}
E[\textbf{run length}^2] &= \sum_{n=1}^{\infty} n^2 \times P(\textbf{run length} = n) \\
&= p_{01}^{10} + \sum_{n=2}^{\infty} n^2 p_{01}^{11} (p_{11}^{11})^{n-2} p_{11}^{10} \\
&= p_{01}^{10} + \frac{p_{01}^{11}}{(p_{11}^{11})^2} \sum_{n=2}^{\infty} n^2 (p_{11}^{11})^n p_{11}^{10} \\
&= p_{01}^{10} + \frac{p_{01}^{11}}{(p_{11}^{11})^2} \left[\sum_{n=0}^{\infty} n^2 (1 - p_{11}^{10})^n p_{11}^{10} - p_{11}^{11} p_{11}^{10}\right] \\
&= p_{01}^{10} + \frac{p_{01}^{11}}{\left(p_{11}^{11}\right)^2} \left[\frac{p_{11}^{11}(2 - p_{11}^{10})}{(p_{11}^{10})^2} - p_{11}^{11} p_{11}^{10}\right] \\
&= p_{01}^{10} + \frac{p_{01}^{11}\left(2 - p_{11}^{10} - \left(p_{11}^{10}\right)^3\right)}{p_{11}^{11}\left(p_{11}^{10}\right)^2}
\end{aligned}$$

since $E[X^2] = \sum_x x^2 P(X = x) = \sum_x x^2 (1-p)^x p = \frac{(1-p)(2-p)}{p^2}$ for a geometric distribution with $P(X = x) = (1-p)^x p$ and $x = 0, 1, 2, 3, ....$ □

Further, the squared expectation of the run length distribution for word lengths $m \geq 3$ is shown in Theorem 4.8. Again, we can find the variance by substituting this in to the equation, Var[**run length**] = $E[\textbf{run length}^2] - (E[\textbf{run length}])^2$, and where $E[\textbf{run length}]$ is found from Theorem 4.6.

**Theorem 4.8** (Squared Expectation of Run Length, $m \geq 3$)**.**

$$
\begin{aligned}
&E\left[\textbf{run length}^2\right] \\
&= \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
&\quad + \sum_{k=2}^{m-1} k^2 \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{k-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=k-1}} \right] \right] \\
&\quad + \frac{1}{\left(p_{2^m-1}^{2^m-1}\right)^m} \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] \right] \\
&\qquad \times \left[ \frac{p_{2^m-1}^{2^m-1}(2-p_{2^m-1}^{2^m-2})}{(p_{2^m-1}^{2^m-2})^2} - \sum_{j=1}^{m-1} j^2 \left(p_{2^m-1}^{2^m-1}\right)^j p_{2^m-1}^{2^m-2} \right].
\end{aligned}
$$

*Proof.*

$$
\begin{aligned}
&E\left[\textbf{run length}^2\right] = \sum_{n=1}^{\infty} n^2 \times P(\textbf{run length} = n) \\
&= \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
&\quad + \sum_{k=2}^{m-1} k^2 \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{k-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=k-1}} \right] \right] \\
&\quad + \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] \right] \\
&\qquad \times \left[ \sum_{n=m}^{\infty} n^2 \left(p_{2^m-1}^{2^m-1}\right)^{n-m} p_{2^m-1}^{2^m-2} \right] \\
&= \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
&\quad + \sum_{k=2}^{m-1} k^2 \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{k-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=k-1}} \right] \right] \\
&\quad + \frac{1}{\left(p_{2^m-1}^{2^m-1}\right)^m} \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] \right] \\
&\qquad \times \left[ \sum_{n=0}^{\infty} n^2 \left(p_{2^m-1}^{2^m-1}\right)^{n} p_{2^m-1}^{2^m-2} - \sum_{j=1}^{m-1} j^2 \left(p_{2^m-1}^{2^m-1}\right)^{j} p_{2^m-1}^{2^m-2} \right] \\
&= \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
&\quad + \sum_{k=2}^{m-1} k^2 \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{k-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=k-1}} \right] \right] \\
&\quad + \frac{1}{\left(p_{2^m-1}^{2^m-1}\right)^m} \left[ \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] \right] \\
&\qquad \times \left[ \frac{p_{2^m-1}^{2^m-1}(2 - p_{2^m-1}^{2^m-2})}{(p_{2^m-1}^{2^m-2})^2} - \sum_{j=1}^{m-1} j^2 \left(p_{2^m-1}^{2^m-1}\right)^{j} p_{2^m-1}^{2^m-2} \right]
\end{aligned}
$$

where,

$$
\pi(i) = \sum_{j=0}^{2^m-1} \prod_{k=0}^{m-3} \left[ p_{2^k(j \bmod 2^{m-k})+\sum_{s=1}^{k} 2^{k-s}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]}^{2^{k+1}(j \bmod 2^{m-k-1})+\sum_{s=1}^{k+1} 2^{k-s+1}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]} \right] \pi(j)
$$

since $E[X^2] = \sum_x x^2 P(X = x) = \sum_x x^2 (1-p)^x p = \frac{(1-p)(2-p)}{p^2}$ for a geometric distribution with $P(X = x) = (1-p)^x p$ and $x = 0, 1, 2, 3, ....$ □

### 4.2.4 Generating Functions

Generating functions (Johnson et al., 2005; Wilf, 1994) are a powerful tool which are often used as an alternative to a distribution when it is too complex to work with. If there is an unknown sequence of numbers (or letters) and there does not exist a simple formula to describe it, then we can employ generating functions. Such functions provide a formula for the sum of a power series where the coefficients of each term make up this original sequence. Often, generating functions are formed from recurrence relationships which give the general form for the sequence, but not a specific formula. This is how I will be forming the generating functions in the remainder of this section, by treating the de Bruijn process run length distribution as an unknown sequence. Although I have defined the formula for the distribution here, it is often the case that the generating functions are easier to work with than the distributions themselves. Since there is a relationship between the de Bruijn run length distribution and a geometric distribution, both the probability and moment generating functions for the geometric distribution are initially given.

First, lets consider a simple example (Wilf, 1994). We have the sequence; $a_0, a_1, a_3, ... = 0, 1, 3, 7, 15, 31, ...$ where we do not know the general formula which generates the sequence, but we do know that the recurrence formula satisfies the following:

$$a_{n+1} = 2a_n + 1 \qquad \text{for } n \geq 0 \text{ and } a_0 = 0. \tag{4.3}$$

We want to find a general form for each term $a_n$ using the generating function $G(x) = \sum_{n\geq 0} a_n x^n$. We will be able to determine each term in the sequence by expanding $G$ into a power series and reading off the coefficients of each term.

To find the generating function, we first multiply both sides of Equation (4.3) by $x^n$ and sum over values of $n$. We then simplify the expression until we are left with

an equation for $G$. This is shown below:

$$\begin{aligned}
\sum_{n\geq 0} a_{n+1}x^n &= \sum_{n\geq 0}(2a_n+1)x^n, \\
a_1 + a_2x + a_3x^2 + ... &= 2\sum_{n\geq 0} a_nx^n + \sum_{n\geq 0} x^n, \\
\{(a_0 + a_1x + a_2x^2 + ...) - a_0\}/x &= 2G(x) + \sum_{n\geq 0} x^n, \\
G(x)/x &= 2G(x) + \frac{1}{1-x}, \\
G(x) &= \frac{x}{(1-x)(1-2x)}.
\end{aligned}$$

Then if we expand $G$ into a series, we can find $a_n$ as the coefficient of $x^n$.

### Probability Generating Function

The probability generating function for the run length distribution takes the form $G(x) = E[x^n] = \sum_{n=1}^{\infty} x^n P(\textbf{run length} = n)$. Probability generating functions are used as a power series representation of the probability mass function. The probability mass function is obtained by taking derivatives of the generating function, such that, $P(X = s) = \frac{G^{(s)}(0)}{s!}$. In other words, the probability of a run length, $s$, will be found by evaluating the $s^{\text{th}}$ derivative of $G$ at zero and dividing by $s$ factorial. For each of the following generating functions, if we expand the result into a power series, we could reproduce the corresponding distribution. The probability generating function for the geometric distribution is given in Theorem 4.9.

**Theorem 4.9** (Geometric Probability Generating Function)**.**

$$G(x) = \frac{p}{1 - x(1-p)}$$

for success probability, $0 \geq p \geq 1$

*Proof.*

$$P(X = n) = (1-p)^n p \qquad \text{for} \quad n \geq 0$$

The recurrence relationship has the following form:

$$\begin{aligned} a_0 &= p \\ a_1 &= (1-p)p \\ a_{n+1} &= (1-p)a_n \qquad \text{for} \quad n \geq 0 \end{aligned}$$

We solve this to find the generating function $G(x) = \sum_{n\geq 0} a_n x^n$. Then, multiplying by $x^n$ and summing over $n$ gives the following:

$$\begin{aligned} \sum_{n\geq 0} a_{n+1}x^n &= \sum_{n\geq 0}(1-p)a_n x^n \\ \Big[a_1 + a_2 x + a_3 x^2 + \ldots\Big] &= (1-p)\sum_{n\geq 0} a_n x^n \\ \frac{1}{x}\Big[\Big(a_0 + a_1 x + a_2 x^2 + \ldots\Big) - a_0\Big] &= (1-p)G(x) \\ \frac{G(x)-p}{x} &= (1-p)G(x) \\ G(x) &= \frac{p}{1-x(1-p)} \end{aligned}$$

□

The probability generating function for the de Bruijn process run length when $m = 2$ is given in Lemma 4.10. Due to the similarities with the geometric distribution, the format will be very similar and we can use some of the same techniques used previously to obtain the final result.

**Lemma 4.10** (Run Length Probability Generating Function, $m = 2$)**.**

$$G(x) = \frac{(p_{01}^{11}p_{11}^{10} - p_{01}^{10}p_{11}^{11})\,x^2 + p_{01}^{10}x}{1 - p_{11}^{11}x}$$

*Proof.*

$$P(\textbf{run length} = n) = \begin{cases} p_{01}^{10} & \text{for } n = 1 \\ p_{01}^{11}(p_{11}^{11})^{n-2}p_{11}^{10} & \text{for } n \geq 2, \end{cases}$$

The recurrence relationship has the following form:

$$
\begin{aligned}
a_0 &= 0 \\
a_1 &= p_{01}^{10} \\
a_2 &= p_{01}^{11} p_{11}^{10} \\
a_{n+1} &= p_{11}^{11} a_n \qquad \text{for} \quad n \geq 2
\end{aligned}
$$

This is solved to find the generating function $G(x) = \sum_{n \geq 0} a_n x^n$. Then, multiplying by $x^n$ and summing over $n$ gives the following:

$$
\begin{aligned}
\sum_{n \geq 2} a_{n+1} x^n &= \sum_{n \geq 2} p_{11}^{11} a_n x^n \\
\left( a_3 x^2 + a_4 x^3 + a_5 x^4 + \ldots \right) &= p_{11}^{11} \left( \sum_{n \geq 0} a_n x^n - a_0 - a_1 x \right) \\
\frac{1}{x} \left[ \left( a_0 + a_1 x + a_2 x^2 + \ldots \right) - a_0 - a_1 x - a_2 x^2 \right] &= p_{11}^{11} \left( G(x) - p_{01}^{10} x \right) \\
\frac{G(x) - p_{01}^{10} x - p_{01}^{11} p_{11}^{10} x^2}{x} &= p_{11}^{11} \left( G(x) - p_{01}^{10} x \right) \\
G(x) &= \frac{\left( p_{01}^{11} p_{11}^{10} - p_{01}^{10} p_{11}^{11} \right) x^2 + p_{01}^{10} x}{1 - p_{11}^{11} x}
\end{aligned}
$$

□

Finally, the probability generating function for the de Bruijn process run length for $m \geq 3$ is given in Theorem 4.11. Again, it has the same structure as both the geometric and $m = 2$ cases, with the slight difference that there is now an additional polynomial added to the final answer. This additional polynomial comes from the initial burn-in period until we hit the point when we reach the all 1 word. Once we have hit this word, and continue adding 1's to the chain, then this becomes equivalent to the geometric distribution. This can be seen in the result since the first polynomial terms represents the burn-in and the fraction term is of the same form as the geometric generating function.

**Theorem 4.11** (Run Length Probability Generating Function, $m \geq 3$)**.**

$$G(x) = \sum_{s=0}^{m} a_s x^s + \frac{p_{2^m-1}^{2^m-1} a_m x^{m+1}}{1 - p_{2^m-1}^{2^m-1} x}$$

where,

$$\begin{aligned}
a_1 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
a_2 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} p_{2^3(i \bmod 2^{m-3})+(2^2-1)}^{2^4(i \bmod 2^{m-4})+(2^3-1)-1} \\
&\vdots \\
a_{m-1} &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=n-1}} \right] \\
a_m &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] p_{2^m-1}^{2^m-2}
\end{aligned}$$

*Proof.*

$$P(\textbf{run length} = n)$$

$$= \begin{cases}
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} & \text{for } n = 1 \\
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} & \text{for } n = 2 : m-1 \\
\qquad \times \left[ \prod_{j=1}^{s-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=s-1}} \right] & \\
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} & \\
\qquad \times \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] & \text{for } n \geq m. \\
\qquad \times \left[ \left( p_{2^m-1}^{2^m-1} \right)^{n-m} p_{2^m-1}^{2^m-2} \right] &
\end{cases}$$

where,

$$\pi(i) = \sum_{j=0}^{2^m-1} \prod_{k=0}^{m-3} \left[ p_{2^k(j \bmod 2^{m-k})+\sum_{w=1}^{k} 2^{k-w}[(\frac{1}{2^{m-w-2}}(i-(i \bmod 2^{m-w-2})))\bmod 2]}^{2^{k+1}(j \bmod 2^{m-k-1})+\sum_{w=1}^{k+1} 2^{k-w+1}[(\frac{1}{2^{m-w-2}}(i-(i \bmod 2^{m-w-2})))\bmod 2]} \right] \pi(j)$$

The recurrence relationship has the following form:

$$
\begin{aligned}
a_0 &= 0 \\
a_1 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
a_2 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} p_{2^3(i \bmod 2^{m-3})+(2^2-1)}^{2^4(i \bmod 2^{m-4})+(2^3-1)-1} \\
&\vdots \\
a_m &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] p_{2^m-1}^{2^m-2} \\
a_{n+1} &= p_{2^m-1}^{2^m-1} a_n \qquad \text{for} \quad n \geq m
\end{aligned}
$$

This is solved to find the generating function $G(x) = \sum_{n \geq 0} a_n x^n$. Then, multiplying by $x^n$ and summing over $n$ gives the following:

$$\sum_{n\geq m} a_{n+1}x^n = \sum_{n\geq m} p_{2^m-1}^{2^m-1} a_n x^n$$

$$\begin{aligned}
\Big(a_{m+1}x^m + a_{m+2}x^{m+1} & \\
+a_{m+3}x^{m+2} + \ldots\Big) &= p_{2^m-1}^{2^m-1}\Bigg(\sum_{n\geq 0} a_n x^n - a_0 - a_1 x - \ldots \\
& \qquad - a_{m-1}x^{m-1}\Bigg) \\
\frac{1}{x}\Big[\Big(a_0 + a_1 x + a_2 x^2 + \ldots\Big) & \\
-a_0 - a_1 x - \ldots - a_m x^m\Big] &= p_{2^m-1}^{2^m-1}\left(G - \sum_{s=0}^{m-1} a_s x^s\right) \\
\frac{G - \sum_{t=0}^{m} a_t x^t}{x} &= p_{2^m-1}^{2^m-1}\left(G - \sum_{s=0}^{m-1} a_s x^s\right) \\
G - \sum_{t=0}^{m} a_t x^t &= x p_{2^m-1}^{2^m-1} G - x p_{2^m-1}^{2^m-1} \sum_{s=0}^{m-1} a_s x^s \\
G\left(x p_{2^m-1}^{2^m-1} - 1\right) &= x p_{2^m-1}^{2^m-1} \sum_{s=0}^{m} a_s x^s - \sum_{t=0}^{m} a_t x^t - x p_{2^m-1}^{2^m-1} a_m x^m \\
G &= \frac{\left(x p_{2^m-1}^{2^m-1} - 1\right)\left(\sum_{s=0}^{m} a_s x^s\right) - x p_{2^m-1}^{2^m-1} a_m x^m}{p_{2^m-1}^{2^m-1} x - 1} \\
G &= \sum_{s=0}^{m} a_s x^s + \frac{p_{2^m-1}^{2^m-1} a_m x^{m+1}}{1 - p_{2^m-1}^{2^m-1} x}
\end{aligned}$$

where,

$$\begin{aligned}
a_1 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
a_2 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} p_{2^3(i \bmod 2^{m-3})+(2^2-1)}^{2^4(i \bmod 2^{m-4})+(2^3-1)-1} \\
&\vdots \\
a_{m-1} &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[\prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=m-2}}\right] \\
a_m &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[\prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)}\right] p_{2^m-1}^{2^m-2}
\end{aligned}$$

□

### Moment Generating Function

As well as probability generating functions, it is also important to calculate moment generating functions. They take the form $G(x) = E[e^{nx}] = \sum_{n=1}^{\infty} e^{nx} P(\textbf{run length} = n)$ for $n = 0, 1, 2, ...$, and are used to find the moments of the distribution. $G(0)$ always exists for the generating function and is equal to one. The moments, $m_s$, are found by taking derivatives of a power series of the generating function, such that, $m_s = G^{(s)}(0)$. In other words, the $s^{\text{th}}$ moment, will be found by evaluating the $s^{\text{th}}$ derivative of $G$ at zero.

As before, I will start by showing the moment generating function for the geometric distribution, given in Theorem 4.12. Results from this theorem will then be used to help form $G(x)$ for the run length distribution.

**Theorem 4.12** (Geometric Moment Generating Function)**.**

$$G(x) = \frac{p}{1 - e^x(1-p)}$$

*Proof.*

$$P(X = n) = (1-p)^n p \qquad \text{for} \quad n \geq 0$$

The recurrence relationship has the following form:

$$\begin{aligned} a_0 &= p \\ a_1 &= (1-p)p \\ a_{n+1} &= (1-p)a_n \qquad \text{for} \quad n \geq 0 \end{aligned}$$

This is solved to find the generating function $G(x) = \sum_{n\geq 0} a_n e^{nx}$. Then, multiplying by $e^{nx}$ and summing over $n$ gives the following:

$$\begin{aligned}
\sum_{n\geq 0} a_{n+1}e^{nx} &= \sum_{n\geq 0}(1-p)a_n e^{nx} \\
\left[a_1 + a_2 e^x + a_3 e^{2x} + \ldots\right] &= (1-p)\sum_{n\geq 0} a_n e^{nx} \\
\frac{1}{e^x}\left[\left(a_0 + a_1 e^x + a_2 e^{2x} + \ldots\right) - a_0\right] &= (1-p)G(x) \\
\frac{G(x)-p}{e^x} &= (1-p)G(x) \\
G(x) &= \frac{p}{1-e^x(1-p)}
\end{aligned}$$

□

The moment generating function for the de Bruijn process run length when $m = 2$ is given in Lemma 4.13. We can clearly see the similarities between this, the geometric moment generating function and the probability generating function for the $m = 2$ de Bruijn run length. If we differentiate this result, we can obtain the results for the expected run length from Lemma 4.3, which is given in Appendix A. Using a computer package such as Maple, we can further obtain the result for the squared expected run length in Lemma 4.7.

**Lemma 4.13** (Run Length Moment Generating Function, $m = 2$)**.**

$$G(x) = \frac{\left(p_{01}^{11}p_{11}^{10} - p_{01}^{10}p_{11}^{11}\right)e^{2x} + p_{01}^{10}e^x}{1 - p_{11}^{11}e^x}$$

*Proof.*

$$P(\textbf{run length} = n) = \begin{cases} p_{01}^{10} & \text{for } n = 1 \\ p_{01}^{11}(p_{11}^{11})^{n-2}p_{11}^{10} & \text{for } n \geq 2, \end{cases}$$

The recurrence relationship has the following form:

$$
\begin{aligned}
a_0 &= 0 \\
a_1 &= p_{01}^{10} \\
a_2 &= p_{01}^{11} p_{11}^{10} \\
a_{n+1} &= p_{11}^{11} a_n \qquad \text{for} \quad n \geq 2
\end{aligned}
$$

This is solved to find the generating function $G(x) = \sum_{n\geq 0} a_n e^{nx}$. Then, multiplying by $e^{nx}$ and summing over $n$ gives the following:

$$
\begin{aligned}
\sum_{n\geq 2} a_{n+1} e^{nx} &= \sum_{n\geq 2} p_{11}^{11} a_n e^{nx} \\
\left(a_3 e^{2x} + a_4 e^{3x} + a_5 e^{4x} + \ldots\right) &= p_{11}^{11} \left(\sum_{n\geq 0} a_n e^{nx} - a_0 - a_1 e^{x}\right) \\
\frac{1}{e^x}\left[\left(a_0 + a_1 e^x + a_2 e^{2x} + \ldots\right) - a_0 - a_1 e^x - a_2 e^{2x}\right] &= p_{11}^{11}\left(G(x) - p_{01}^{10} e^x\right) \\
\frac{G(x) - p_{01}^{10} e^x - p_{01}^{11} p_{11}^{10} e^{2x}}{e^x} &= p_{11}^{11}\left(G(x) - p_{01}^{10} e^x\right) \\
G(x) &= \frac{\left(p_{01}^{11} p_{11}^{10} - p_{01}^{10} p_{11}^{11}\right) e^{2x} + p_{01}^{10} e^x}{1 - p_{11}^{11} e^x}
\end{aligned}
$$

□

Finally, the moment generating function for a length $m \geq 3$ de Bruijn process run length is given in Theorem 4.14. As for the case in Theorem 4.11, we can see the similarities to the geometric case, but with the addition of a polynomial term representing the burn-in period to the de Bruijn word consisting of all 1's. The addition of the polynomial is necessary as it represents all of the different possible starting points for all of the different runs created until we reach long runs of 1's. Using Maple, we can obtain the results for the expected run length and squared expected run length shown in Lemmas 4.6 and 4.8 from the moment generating function.

**Theorem 4.14** (Run Length Moment Generating Function, $m \geq 3$)**.**

$$G(x) = \sum_{s=0}^{m} a_s e^{sx} + \frac{p_{2^m-1}^{2^m-1} a_m e^{(m+1)x}}{1 - p_{2^m-1}^{2^m-1} e^x}$$

where,

$$\begin{aligned}
a_1 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
a_2 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} p_{2^3(i \bmod 2^{m-3})+(2^2-1)}^{2^4(i \bmod 2^{m-4})+(2^3-1)-1} \\
&\vdots \\
a_{m-1} &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=n-1}} \right] \\
a_m &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] p_{2^m-1}^{2^m-2}
\end{aligned}$$

*Proof.*

$$P(\textbf{run length} = n)$$

$$= \begin{cases}
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} & \text{for } n = 1 \\
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} & \text{for } n = 2 : m-1 \\
\qquad \times \left[ \prod_{j=1}^{n-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=n-1}} \right] & \\
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} & \\
\qquad \times \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] & \text{for } n \geq m. \\
\qquad \times \left[ \left( p_{2^m-1}^{2^m-1} \right)^{n-m} p_{2^m-1}^{2^m-2} \right] &
\end{cases}$$

where,

$$\pi(i) = \sum_{j=0}^{2^m-1} \prod_{k=0}^{m-3} \left[ p_{2^k(j \bmod 2^{m-k})+\sum_{s=1}^{k} 2^{k-s}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]}^{2^{k+1}(j \bmod 2^{m-k-1})+\sum_{s=1}^{k+1} 2^{k-s+1}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]} \right] \pi(j)$$

The recurrence relationship has the following form:

$$
\begin{aligned}
a_0 &= 0 \\
a_1 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
a_2 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} p_{2^3(i \bmod 2^{m-3})+(2^2-1)}^{2^4(i \bmod 2^{m-4})+(2^3-1)-1} \\
&\vdots \\
a_{n+1} &= p_{2^m-1}^{2^m-1} a_n \qquad \text{for} \quad n \geq m
\end{aligned}
$$

This is solved to find the generating function $G(x) = \sum_{n\geq 0} a_n e^{nx}$. Then, multiplying by $e^{nx}$ and summing over $n$ gives the following:

$$\sum_{n\geq m} a_{n+1}e^{nx} = \sum_{n\geq m} p_{2^m-1}^{2^m-1} a_n e^{nx}$$

$$\begin{aligned}
\Big(a_{m+1}e^{mx} + a_{m+2}e^{(m+1)x} & \\
+a_{m+3}e^{(m+2)x} + \dots\Big) &= p_{2^m-1}^{2^m-1}\Bigg(\sum_{n\geq 0} a_n e^{nx} - a_0 - a_1 e^x - \dots \\
&\qquad - a_{m-1}e^{(m-1)x}\Bigg) \\
\frac{1}{e^x}\Big[\Big(a_0 + a_1 e^x + a_2 e^{2x} + \dots\Big) & \\
-a_0 - a_1 e^x - \dots - a_m e^{mx}\Big] &= p_{2^m-1}^{2^m-1}\left(G - \sum_{s=0}^{m-1} a_s e^{sx}\right) \\
\frac{G - \sum_{t=0}^{m} a_t e^{tx}}{e^x} &= p_{2^m-1}^{2^m-1}\left(G - \sum_{s=0}^{m-1} a_s e^{sx}\right) \\
G - \sum_{t=0}^{m} a_t e^{tx} &= e^x p_{2^m-1}^{2^m-1} G - e^x p_{2^m-1}^{2^m-1} \sum_{s=0}^{m-1} a_s e^{sx} \\
G\left(e^x p_{2^m-1}^{2^m-1} - 1\right) &= e^x p_{2^m-1}^{2^m-1} \sum_{s=0}^{m} a_s e^{sx} - \sum_{t=0}^{m} a_t e^{tx} - e^x p_{2^m-1}^{2^m-1} a_m e^{mx} \\
G &= \frac{\left(e^x p_{2^m-1}^{2^m-1} - 1\right)\left(\sum_{s=0}^{m} a_s e^{sx}\right) - e^x p_{2^m-1}^{2^m-1} a_m e^{mx}}{p_{2^m-1}^{2^m-1} e^x - 1} \\
G &= \sum_{s=0}^{m} a_s e^{sx} + \frac{p_{2^m-1}^{2^m-1} a_m e^{(m+1)x}}{1 - p_{2^m-1}^{2^m-1} e^x}
\end{aligned}$$

where,

$$\begin{aligned}
a_1 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} \\
a_2 &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} p_{2^3(i \bmod 2^{m-3})+(2^2-1)}^{2^4(i \bmod 2^{m-4})+(2^3-1)-1} \\
&\vdots \\
a_{m-1} &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[\prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=n-1}}\right] \\
a_m &= \sum_{i=0}^{2^{m-2}-1} \pi(i) p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[\prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)}\right] p_{2^m-1}^{2^m-2}
\end{aligned}$$

□

### Cumulants

Cumulants (McCullagh, 1987; Wilf, 1994) offer an alternative to moment generating functions, and the first three are equal to the central moments. The first cumulant is the mean, the second is the variance and the third is the third central moment. Cumulants are often used instead of moments, or even instead of the distribution itself, because they are simpler to work with, especially if the form of the distribution is complicated. The cumulant generating function is found by taking the log of the moment-generating function such that:

$$\begin{aligned} K(x) &= \log\left[G(x)\right] \\ &= \log\left[E(e^{nx})\right] \\ &= \log\left[\sum_{n=1}^{\infty} e^{nx} P(\textbf{run length} = n)\right] \end{aligned}$$

The cumulants are found by taking derivatives of a power series expansion of the cumulant generating function, and then evaluating at zero. So, the $s$th cumulant, $\mu_s$, is found by evaluating $K^s(0)$. Due to the relationship between the culmulant and moment generating functions, it is therefore possible to obtain the cumulants from the moments and the moments from the cumulants.

When trying to find the cumulant generating function for the run length distribution, we take the natural log of the moment generating function. However, by doing this we find that the expression ends up more complicated due to the term, $1 - p_{2^m-1}^{2^m-1}$, in the denominator which is hard to simplify. Therefore, I choose to not proceed any further. I will, however, briefly use the cumulant generating functions in Section 4.2.6 for examples of the run length distribution. I will use the fourth central moment, the kurtosis, $\mu_4 = K^4(0)$, to find the variance of the sample variance, $\frac{\mu_4}{n} - \frac{\sigma^4(n-3)}{n(n-1)}$.

It is possible to develop new cumulants. For example there exist tree cumulants, binary cumulants (Zwiernik and Smith, 2012) and generalised cumulants (McCullagh, 1987). Tree cumulants are used for describing models that have a specific graphical structure, whereas binary cumulants are used for simplifying distributions on binary sequences. Generalised cumulants are formed from different groups of joint cumulants

which can be partitioned in different ways based on different correlations between variables. It therefore may be possible to generate a new type of cumulant that would be more suited to de Bruijn graphs, but I leave this for future exploration.

### 4.2.5 Non-Stationary de Bruijn processes

Non-stationary models were briefly mentioned in Chapter 3 and, although I do not go into great detail on the subject, I will show how non-stationarity can change the run length distribution and expected run lengths of length $m = 2$ de Bruijn processes.

For the non-stationary case, the transition matrix of the latent Markov chain changes with respect to time, such that the transition probabilities are different at each change in state. Hence, for a length two de Bruijn process, the non-stationary transition matrix is as follows:

$$T^{(t)} = \begin{pmatrix} 1 - p_{00}^{01(t)} & p_{00}^{01(t)} & 0 & 0 \\ 0 & 0 & 1 - p_{01}^{11(t)} & p_{01}^{11(t)} \\ 1 - p_{10}^{01(t)} & p_{10}^{01(t)} & 0 & 0 \\ 0 & 0 & 1 - p_{11}^{11(t)} & p_{11}^{11(t)} \end{pmatrix},$$

for $t = 0, 1, 2, ....$

The distribution of the run length for word length two is given in Lemma 4.15. This is equivalent to the stationary version shown in Lemma 4.1 with the addition of the transition probabilities being dependent on the time $t$. The time starts with $t = 1$, representing the beginning of the run of 1's, and finishes when the 0 is added to complete the run.

**Lemma 4.15** (Non-Stationary Run Length Distribution, $m = 2$)**.**

$$P(\textbf{run length} = n) = \begin{cases} p_{01}^{10(t=1)} & \text{for } n = 1 \\ p_{01}^{11(t=1)} \left[\prod_{i=2}^{n-1} p_{11}^{11(t=i)}\right] p_{11}^{10(t=n)} & \text{for } n \geq 2, \end{cases}$$

*Proof.* Follows from Lemma 4.3 with the addition of the transition probabilities being dependent on time, $t$. □

Similarly as for the stationary case, the expected run length for the non-stationary length $m = 2$ de Bruijn processes is given in Lemma 4.16. Since the transition probabilities are now dependent on the time step, we can no longer use results from the geometric distribution to simplify results.

**Lemma 4.16** (Non-Stationary Expected Run Length, $m = 2$)**.**

$$E\left[\textbf{run length}\right] = p_{01}^{10(t=1)} + p_{01}^{11(t=1)}\left[\sum_{n=2}^{\infty} n \prod_{i=2}^{n-1} p_{11}^{11(t=i)}\, p_{11}^{10(t=n)}\right]$$

*Proof.*

$$\begin{aligned} E\left[\textbf{run length}\right] &= \sum_{n=1}^{\infty} n \times P(\textbf{run length} = n) \\ &= p_{01}^{10(t=1)} + \sum_{n=2}^{\infty} n p_{01}^{11(t=1)}\left[\prod_{i=2}^{n-1} p_{11}^{11(t=i)}\right] p_{11}^{10(t=n)} \\ &= p_{01}^{10(t=1)} + p_{01}^{11(t=1)}\left[\sum_{n=2}^{\infty} n \prod_{i=2}^{n-1} p_{11}^{11(t=i)}\, p_{11}^{10(t=n)}\right] \end{aligned}$$

□

### 4.2.6 Examples

In this section I will discuss several examples which are designed to compare the de Bruijn process run length distribution against run lengths that are created though simulation. I will be considering the examples shown in Figures 3.4, 3.6 and 3.7 from Chapter 3. These figures consisted of six examples from de Bruijn processes with word length $m = 2$ with the transition probabilities, $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\}$, being: $\{0.5, 0.5, 0.5, 0.5\}$, $\{0.25, 0.75, 0.25, 0.75\}$, $\{0.1, 0.9, 0.1, 0.9\}$, $\{0.05, 0.95, 0.05, 0.95\}$, $\{0.9, 0.1, 0.9, 0.1\}$ and $\{0.775, 0.8, 0.8, 0.9\}$ respectively.

Table 4.1 shows the probabilities of getting a run length of $n$ 1's, where $n = 1, ..., 10$, from the run length distribution in Theorem 4.2 for these six examples (presented in the same order as above so that DBP 1 refers to the de Bruijn process with transition probabilities $\{0.5, 0.5, 0.5, 0.5\}$ and DBP 6 refers to probabilities $\{0.775, 0.8, 0.8, 0.9\}$).

| Run Length, n | DBP 1 | DBP 2 | DBP 3 | DBP 4 | DBP 5 | DBP 6 |
|---|---|---|---|---|---|---|
| 1 | 0.5 | 0.25 | 0.1 | 0.05 | 0.9 | 0.2 |
| 2 | 0.25 | 0.188 | 0.09 | 0.0475 | 0.09 | 0.08 |
| 3 | 0.125 | 0.141 | 0.081 | 0.0451 | 0.009 | 0.072 |
| 4 | 0.0625 | 0.105 | 0.0729 | 0.0429 | 0.0009 | 0.0648 |
| 5 | 0.0313 | 0.0791 | 0.0656 | 0.0407 | $9 \times 10^{-4}$ | 0.0583 |
| 6 | 0.0156 | 0.0593 | 0.0590 | 0.0387 | $9 \times 10^{-5}$ | 0.0525 |
| 7 | 0.00781 | 0.0445 | 0.0531 | 0.0368 | $9 \times 10^{-6}$ | 0.0472 |
| 8 | 0.00391 | 0.0334 | 0.0478 | 0.0349 | $9 \times 10^{-7}$ | 0.0425 |
| 9 | 0.00195 | 0.0250 | 0.0430 | 0.0332 | $9 \times 10^{-8}$ | 0.0383 |
| 10 | 0.000977 | 0.0188 | 0.0387 | 0.0315 | $9 \times 10^{-9}$ | 0.0344 |

Table 4.1 Table to show the probabilities of getting run lengths of $n = 1, ..., 10$ for six different de Bruijn processes of word length $m = 2$. The corresponding transition probabilities ($\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\}$) for these four processes are as follows, DBP 1: $\{0.5, 0.5, 0.5, 0.5\}$, DBP 2: $\{0.25, 0.75, 0.25, 0.75\}$ , DBP 3: $\{0.1, 0.9, 0.1, 0.9\}$, DBP 4: $\{0.05, 0.95, 0.05, 0.95\}$, DBP 5: $\{0.9, 0.1, 0.9, 0.1\}$, DBP 6: $\{0.775, 0.8, 0.8, 0.9\}$.

We see that DBP 1 has a much higher chance at 50% of a short run length of just one as compared to the next three de Bruijn processes (DBP 2, DBP 3, DBP 4). This then drops for the stickier processes until DBP 4 only has a 5% chance of a short run length. We also notice that the probabilities for the stickier processes stay fairly similar for all run lengths compared to the Bernoulli process (DBP 1) that reduces very quickly. This is what we would expect since the run lengths for independent Bernoulli trials tend to be very short. We also see this behaviour in the anti-sticky example in DBP 5 where the probabilities for a run length any higher than two quickly becomes very small. There is a 90% chance of a run length of just a single 1 since this process is designed to alternate between the two letters. The non-symmetric example in DBP 6 is shown to behave fairly similar to either DBP 2 or DBP 3.

We will now look at comparing the theoretical expected run lengths of 1's and the average run lengths of 1's calculated from the simulations in Chapter 3. In the same order as above, the average run lengths from the simulations were: 2.02, 4.10, 9.47, 19.60, 1.11 and 9.09 for length 200 sequences. Using Equation (4.3), the calculated theoretical expected run lengths for the de Bruijn processes are: 2, 4, 10, 20, 1.11 and 9 respectively (see Table 4.3). For all cases, both values are shown to agree

with on average a difference of 2.02%. Hence, this gives confidence that both the simulated and analytic run lengths are calculated correctly.

If we now compare the theoretical variance of run length with the simulated variance of run length we can see that the error between the results is much higher. There is likely to be more variance in the simulations, especially if the de Bruijn Markov chains are not run for a sufficient amount of time. For the example above, the variance of run lengths from the simulations were: 2.21, 12.07, 93.57, 304.76, 0.12 and 88.69 respectively. We can see that when the de Bruijn process gets very sticky, the resulting variances become much larger with a larger difference even between DBP 3 and DBP 4 where the transition probabilities are fairly similar. This can also be seen in the histograms in Figure 3.5, where the spread of run lengths gets much larger in the sticky processes. We can see that occasionally we get a very long run length, but also get times when we have very short lengths. If we compare this to the analytical variances calculated, we get: 2, 12, 90, 380, 0.12 and 88 respectively (see Table 4.3). So, for the first three de Bruijn processes we get very similar results, but for the very sticky case (DBP 4), the variances from the simulation are lower. This is likely to be due to the simulations being able to occasionally produce a very long run length which will consequently skew the results by a large amount. We may also not be running the de Bruijn process for a sufficient amount of time. Table 4.3 shows the analytical and simulated expected run lengths and variance of run length for eight de Bruijn process examples. Alongside this are values for two standard deviations of the sample expected run length and two standard deviations of the sample variance of run length. These are calculated using the expressions, $\sqrt{\frac{\sigma^2}{n}}$ and $\sqrt{\frac{\mu_4}{n} - \frac{\sigma^4(n-3)}{n(n-1)}}$ respectively, where $n = 200$ and $\mu_4$ is the kurtosis. This is the fourth central moment and is found from the fourth differential of the cumulant generating function of the run length distribution evaluated at zero (see Section 4.2.4). We can see here that although the analytical and simulated variances of the DBP 4 example do not agree, the difference is within $\pm$ two standard deviations.

The expected run length of the structured example (DBP 5) is 1.11, which is what we would expect since the de Bruijn process is designed to be constantly flipping

letters. The variance for this process is 0.12 which confirms that the process is designed to be very structured and does not alter much from having an average run length close to one. These both match exactly to the theoretical results. For the non-symmetrical example (DBP 6) the expected run length is 9.09 and the variance is 88.69. This is to be expected as I have defined the process to have the same stickiness to 1's as DBP 3 with transition probabilities $\{0.1, 0.9, 0.1, 0.9\}$. As stated in Chapter 3, I have defined the stickiness to be the same, but the example differs due the marginal probabilities on the letters ($\pi(\{0\}) = 0.2$ and $\pi(\{1\}) = 0.8$). So, the run lengths of 1's should be similar, but the number of 0's is reduced to just 20% of the overall number of letters.

In Chapter 3, I also showed in Example 3.9 how certain de Bruijn processes with higher word lengths can produce equivalent chains of 0's and 1's to those with shorter word lengths. Hence, some larger word length de Bruijn processes can collapse down to a shorter word length process. Table 4.2 gives the probabilities of getting a run length of $n$ 1's for $n = 1, ..., 4$ for three different de Bruijn processes with word lengths $m = 2$ (DBP 3), $m = 3$ (DBP 7) and $m = 4$ (DBP 8) respectively. All transition probabilities for each process are such that there is a 90% chance of remaining at the same letter and a 10% chance of transitioning to a different letter. This table emphasizes how each de Bruijn process is equivalent since the probabilities are equal for each $n$.

| Run Length, n | DBP 3 | DBP 7 | DBP 8 |
|---|---|---|---|
| 1 | 0.1 | 0.1 | 0.1 |
| 2 | 0.09 | 0.09 | 0.09 |
| 3 | 0.081 | 0.081 | 0.081 |
| 4 | 0.0729 | 0.0729 | 0.0729 |

Table 4.2 Table to show the probabilities of getting run lengths of $n = 1, ..., 4$ for three equivalent de Bruijn processes of word lengths $m = 2$, $m = 3$ and $m = 4$. The transition probabilities are as follows: DBP 3: $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\} = \{0.1, 0.9, 0.1, 0.9\}$, DBP 7: $\{p_{000}^{001}, p_{001}^{011}, p_{010}^{101}, p_{011}^{111}, p_{100}^{001}, p_{101}^{011}, p_{110}^{101}, p_{111}^{111}\} = \{0.1, 0.9, 0.1, 0.9, 0.1, 0.9, 0.1, 0.9\}$, DBP 8: $\{p_{0000}^{0001}, p_{0001}^{0011}, p_{0010}^{0101}, p_{0011}^{0111}, p_{0100}^{1001}, p_{0101}^{1011}, p_{0110}^{1101}, p_{0111}^{1111}, p_{1000}^{0001}, p_{1001}^{0011}, p_{1010}^{0101}, p_{1011}^{0111}, p_{1100}^{1001}, p_{1101}^{1011}, p_{1110}^{1101}, p_{1111}^{1111}\} = \{0.1, 0.9, 0.1, 0.9, 0.1, 0.9, 0.1, 0.9, 0.1, 0.9, 0.1, 0.9, 0.1, 0.9, 0.1, 0.9\}$

| D. B. Process | A. Exp | S. Exp | 2 Sd(S. E.) | A. Var | S. Var | 2 Sd(S. V.) |
|---|---|---|---|---|---|---|
| DBP 1 | 2 | 2.02 | 0.20 | 2 | 2.21 | 0.66 |
| DBP 2 | 4 | 4.10 | 0.49 | 12 | 12.07 | 3.83 |
| DBP 3 | 10 | 9.47 | 1.34 | 90 | 93.57 | 28.52 |
| DBP 4 | 20 | 19.60 | 2.76 | 380 | 304.76 | 120.32 |
| DBP 5 | 1.11 | 1.11 | 0.05 | 0.12 | 0.12 | 0.063 |
| DBP 6 | 9 | 9.09 | 1.33 | 88 | 88.69 | 29.10 |
| DBP 7 | 10 | 10.25 | 1.34 | 90 | 95.05 | 28.52 |
| DBP 8 | 10 | 9.91 | 1.34 | 90 | 93.78 | 28.52 |

Table 4.3 Table to show the analytical expectation (A. Exp), simulated expectation (S. Exp), two standard deviations of the simulated expectation (Var(S. E.)), analytical variance (A. Exp), simulated variance (S. Var) and two standard deviations of the simulated variance (Var(S. V.)) of the run length distribution given a sequence of length 200 for eight different de Bruijn processes. These are the same de Bruijn processes from Tables 4.1 and 4.2.

## 4.3 Inference

In this section I will outline a method for inference such that, given a chain of 0's and 1's we will be able to estimate the de Bruijn process that may have generated it. This will include estimating both the length of the de Bruijn word, $m$, and the transition probabilities, $p$.

I first look to define the word marginal likelihood for a sequence of 0's and 1's at steady state. We will then be able to estimate the marginal distribution for the de Bruijn words in the chain. Either maximum likelihood or Bayesian inference can be used, which should give the same answer as counting the proportion of each of the words in the given sequence. For the de Bruijn structure, the likelihood takes a similar form to a multivariate Bernoulli likelihood. The likelihood of the de Brujin process with word length $m = 2$ (consisting of words $\{00, 01, 10, 11\}$) is given in Lemma 4.17. $\pi(i)$ is the marginal probability for the word $i$, and $n_i$ is the number of words, $i$, in that sequence. The marginal likelihood is the joint distribution of a sequence in terms of the proportion of different words in that given sequence. Therefore, it consists of the marginal probabilities for each word raised to the power of how many there are in the sequence. It is possible to reduce the dimensionality slightly using the fact that all the marginal probabilities sum to 1, and that the $n$'s add up to the total sum of all words (defined as $N$).

**Lemma 4.17** (Marginal Likelihood, $m = 2$)**.**

$$\begin{aligned}\mathcal{L}(seq) &= \pi(00)^{n_{00}}\pi(01)^{n_{01}}\pi(10)^{n_{10}}\pi(11)^{n_{11}} \\ &= \pi(00)^{n_{00}}\pi(01)^{n_{01}}\pi(10)^{n_{10}}[1-(\pi(00)+\pi(01)+\pi(10))]^{N-(n_{00}+n_{01}+n_{10})}\end{aligned}$$

for $N = n_{00} + n_{01} + n_{10} + n_{11}$ and $\pi(00) + \pi(01) + \pi(10) + \pi(11) = 1$

*Proof.* Given a sequence of letters of length $N + 1$, the sequence can be split up into its associated words of length $m = 2$, where $\pi(i)$ gives the probability of obtaining the word $i$. The joint distribution is given by the product of the respective $\pi$'s for each word and $N$ gives the total number of words. By collecting like terms, the above definition is given. □

The marginal likelihood for a de Bruijn process with general word length $m$ is given in Theorem 4.18. This theorem has the same structure as for the $m = 2$ case, where we are also able to take advantage of the $\pi$'s summing to one, and that we know the total number of words in the given sequence. Again, I have chosen to write this in terms of the decimal representation of the binary values.

**Theorem 4.18** (Marginal Likelihood, $m \geq 1$)**.**

$$\begin{aligned}\mathcal{L}(seq) &= \prod_{i=0}^{2^m-1}\pi(i)^{n_i} \\ &= \prod_{i=0}^{2^m-2}\pi(i)^{n_i}\left[1-\left(\sum_{j=0}^{2^m-2}\pi(j)\right)\right]^{N-\left(\sum_{j=0}^{2^m-2}n_j\right)}\end{aligned}$$

for $N = \sum_{j=0}^{2^m-1} n_j$ and $\sum_{j=0}^{2^m-1}\pi(j) = 1$

*Proof.* See Appendix A □

It is then possible to write down the likelihood for the transition probabilities so that they can be estimated given a sequence. To begin with, I assume that the word length is known, and concentrate on estimating the transitions. I note here that it is far more difficult to estimate the word length as it is the word length that controls how many transition parameters there are to be estimated. One possible solution is

to use reversible jump MCMC (Green, 1995), but I have developed a more efficient method explained further down in this section.

The likelihood for the transitions of a length $m = 2$ de Bruijn process is given in Lemma 4.19, and the likelihood for the $m \geq 1$ case is given in Theorem 4.20. The likelihood is the joint distribution of the sequence in terms of the transition probabilities. However, the form of the likelihood does not obviously indicate that this does in fact give the joint distribution of the sequence. The ordering of the letters in the sequence is fixed and, due to the de Bruijn structure, the ordering of the words is very important. This is due to the fact that for each word in the sequence there are only two possible words that can be transitioned to. Hence, the distinct number of times each transition occurs in the sequence defines the exact ordering of the letters. Recall that each term $p_i^j$ in the likelihood is the transition probability from word $i$ to word $j$, and $n_i^j$ is the corresponding number of times this transition occurs in the given sequence.

**Lemma 4.19** (Transition Likelihood, $m = 2$)**.**

$$\begin{aligned}\mathcal{L}(seq|p) &= (p_{00}^{00})^{n_{00}^{00}}\,(p_{00}^{01})^{n_{00}^{01}}\,(p_{01}^{10})^{n_{01}^{10}}\,(p_{01}^{11})^{n_{01}^{11}}\,(p_{10}^{00})^{n_{10}^{00}}\,(p_{10}^{01})^{n_{10}^{01}}\,(p_{11}^{10})^{n_{11}^{10}}\,(p_{11}^{11})^{n_{11}^{11}} \\ &= (1-p_{00}^{01})^{n_{00}^{00}}\,(p_{00}^{01})^{n_{00}^{01}}\,(1-p_{01}^{11})^{n_{01}^{10}}\,(p_{01}^{11})^{n_{01}^{11}}\,(1-p_{10}^{01})^{n_{10}^{00}}\,(p_{10}^{01})^{n_{10}^{01}} \\ &\quad\times(1-p_{11}^{11})^{n_{11}^{10}}\,(p_{11}^{11})^{n_{11}^{11}},\end{aligned}$$

*Proof.* Assume a sequence of letters, $L = l_1, l_2, ..., l_n$, where $l_i \in [0, 1]$ and the ordering is fixed. This sequence can be expressed in terms of its de Bruijn words such that $L = w_1, w_2, ..., w_{n-1}$, where $w_i$ are the de Bruijn words of length $m = 2$. Consider the joint distribution of this sequence. Starting from $w_1$, the probability of transitioning to the next word is $p_{w_1}^{w_2}$. The probability of transitioning to the next following word is, $p_{w_2}^{w_3}$. This is continued until the transition $p_{w_{n-2}}^{w_{n-1}}$ is reached and produces the joint distribution, $\mathcal{L}(L|p) = p_{w_1}^{w_2} \times p_{w_2}^{w_3} \times ... \times p_{w_{n-2}}^{w_{n-1}}$. By collecting like terms for each possible transition probability the above result is given. □

**Theorem 4.20** (Transition Likelihood, $m \geq 1$)**.**

$$\mathcal{L}(seq|p) = \prod_{i=0}^{2^m-1} \left(1 - p_i^{(2i+1)\ mod\ 2^m}\right)^{n_i^{((2i+1)\ mod\ 2^m)-1}} \left(p_i^{(2i+1)\ mod\ 2^m}\right)^{n_i^{((2i+1)\ mod\ 2^m)}}.$$

*Proof.* See Appendix A □

The likelihood can then be used to estimate the transition probabilities given a sequence of 0's and 1's either through frequentist maximum likelihood estimation or using Bayesian methods. Given Bayes' theorem, we know that the posterior distribution is equal to the product of the likelihood, $\mathcal{L}(seq|p)$, and prior, $P(p)$, normalised by the probability of the data, $P(seq)$, to give:

$$\begin{aligned} P(p|seq) &= \frac{\mathcal{L}(seq|p)P(p)}{P(seq)} \\ &= \frac{\mathcal{L}(seq|p)P(p)}{\int \mathcal{L}(seq|p)P(p)dp}. \end{aligned} \tag{4.4}$$

The prior distribution is where we can specify any prior knowledge we might have about the transition probabilities. For example, we may know that the sequence is very sticky to 1's, and we can incorporate this into the prior distribution to put higher weighting onto the transition that remains at the all 1 word. We can also look at the form of the likelihood to help us choose the form of the selected prior distribution. If the posterior distribution is in the same probability distribution family as the prior probability distribution, then they are conjugate and the prior is a conjugate prior for the likelihood function. Looking at the form of the transition likelihood, we can see that it has a very similar structure to the beta distribution. Hence we are able to use a Beta prior of the form, $P(p) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}p^{\alpha-1}(1-p)^{\beta-1}$, for $\alpha > 0$ and $\beta > 0$, to produce the posterior distribution for the transition probabilities.

If we initially start with the de Bruijn word length $m = 2$ case, the transition likelihood is as above:

$$\mathcal{L} = (1-p_{00}^{01})^{n_{00}^{00}}\,(p_{00}^{01})^{n_{00}^{01}}\,(1-p_{01}^{11})^{n_{01}^{10}}\,(p_{01}^{11})^{n_{01}^{11}}\,(1-p_{10}^{01})^{n_{10}^{00}}\,(p_{10}^{01})^{n_{10}^{01}}\,(1-p_{11}^{11})^{n_{11}^{10}}\,(p_{11}^{11})^{n_{11}^{11}},$$

which we can then combine with a Beta prior to obtain a posterior distribution which is proportional to the following:

$$\begin{aligned}
P(p|seq) \propto\ & \mathcal{L}(seq|p)P(p) \\
=\ & (1-p_{00}^{01})^{n_{00}^{00}}\,(p_{00}^{01})^{n_{00}^{01}}\,(1-p_{00}^{01})^{\beta_1-1}\,(p_{00}^{01})^{\alpha_1-1} \\
& \times (1-p_{01}^{11})^{n_{01}^{10}}\,(p_{01}^{11})^{n_{01}^{11}}\,(1-p_{01}^{11})^{\beta_2-1}\,(p_{01}^{11})^{\alpha_2-1} \\
& \times (1-p_{10}^{01})^{n_{10}^{00}}\,(p_{10}^{01})^{n_{10}^{01}}\,(1-p_{10}^{01})^{\beta_3-1}\,(p_{10}^{01})^{\alpha_3-1} \\
& \times (1-p_{11}^{11})^{n_{11}^{10}}\,(p_{11}^{11})^{n_{11}^{11}}\,(1-p_{11}^{11})^{\beta_4-1}\,(p_{11}^{11})^{\alpha_4-1} \\
\\
=\ & (1-p_{00}^{01})^{n_{00}^{00}+\beta_1-1}\,(p_{00}^{01})^{n_{00}^{01}+\alpha_1-1} \\
& \times (1-p_{01}^{11})^{n_{01}^{10}+\beta_2-1}\,(p_{01}^{11})^{n_{01}^{11}+\alpha_2-1} \\
& \times (1-p_{10}^{01})^{n_{10}^{00}+\beta_3-1}\,(p_{10}^{01})^{n_{10}^{01}+\alpha_3-1} \\
& \times (1-p_{11}^{11})^{n_{11}^{10}+\beta_4-1}\,(p_{11}^{11})^{n_{11}^{11}+\alpha_4-1}.
\end{aligned}$$

Hence, the posterior distribution for the de Bruijn process transition probabilities is a product of beta densities. Although the denominator in Equation (4.4) is often difficult to define explicitly, it can be found in this example due to the prior and posterior distributions having a conjugate relationship. Hence, we can state the following:

$$\begin{aligned}
\int P(seq|p)P(p)dp =& \frac{\Gamma(n_{00}^{00}+\beta_1)\Gamma(n_{00}^{01}+\alpha_1)}{\Gamma(n_{00}^{00}+n_{00}^{01}+\beta_1+\alpha_1)} \times \frac{\Gamma(n_{01}^{10}+\beta_2)\Gamma(n_{01}^{11}+\alpha_2)}{\Gamma(n_{01}^{10}+n_{01}^{11}+\beta_2+\alpha_2)} \times \\
& \frac{\Gamma(n_{10}^{00}+\beta_3)\Gamma(n_{10}^{01}+\alpha_3)}{\Gamma(n_{10}^{00}+n_{10}^{01}+\beta_3+\alpha_3)} \times \frac{\Gamma(n_{11}^{10}+\beta_4)\Gamma(n_{11}^{11}+\alpha_4)}{\Gamma(n_{00}^{10}+n_{11}^{11}+\beta_4+\alpha_4)}.
\end{aligned} \tag{4.5}$$

This equation is known as the model evidence. For the general case when $m \geq 1$, the posterior distribution is given in Theorem 4.21.

**Theorem 4.21** (Posterior Distribution for de Bruijn Probability Transitions, $m \geq 1$)**.**

$$\begin{aligned}
P(p|seq) &= \frac{\mathcal{L}(seq|p,m)P(p|m)}{P(seq)} \\
&= \frac{\mathcal{L}(seq|p,m)P(p|m)}{\int \mathcal{L}(seq|p,m)P(p|m)dp}
\end{aligned}$$

where,

$$\mathcal{L}(seq|p,m)P(p|m) = \prod_{i=0}^{2^m-1} (1-p_i^{((2i+1) \bmod 2^m)})^{n_i^{((2i+1) \bmod 2^m)-1}+\beta_{i+1}-1} \times (p_i^{((2i+1) \bmod 2^m)})^{n_i^{((2i+1) \bmod 2^m)}+\alpha_{i+1}-1}$$

and

$$\int P(seq|p,m)P(p|m)dp = \prod_{i=0}^{2^m-1} \frac{\Gamma(n_i^{((2i+1) \bmod 2^m)-1}+\beta_{i+1})\Gamma(n_i^{((2i+1) \bmod 2^m)})+\alpha_{i+1})}{\Gamma(n_i^{((2i+1) \bmod 2^m)-1}+n_i^{((2i+1) \bmod 2^m)}+\beta_{i+1}+\alpha_{i+1})}$$

*Proof.* See Appendix A □

For large data sizes, this result can experience computational problems. Hence, we can either work in the log scale, or we can use the following:

$$\begin{aligned}\frac{\Gamma(n_1+\alpha)\Gamma(n_2+\beta)}{\Gamma(n_1+n_2+\alpha+\beta)} &= \frac{\frac{\alpha\Gamma(\alpha)}{\alpha+n_1}\prod_{i=1}^{n_1}(\alpha+i)\frac{\beta\Gamma(\beta)}{\beta+n_2}\prod_{j=1}^{n_2}(\beta+j)}{\frac{(\alpha+\beta)\Gamma(\alpha+\beta)}{\alpha+\beta+n_1+n_2}\prod_{k=1}^{n_1+n_2}(\alpha+\beta+k)} \\ &= \frac{\alpha\beta\Gamma(\alpha)\Gamma(\beta)(\alpha+\beta+n_1+n_2)}{\Gamma(\alpha+\beta)(\alpha+\beta)(\alpha+n_1)(\beta+n_2)}\prod_{i=1}^{n_1}\frac{\alpha+i}{\alpha+\beta+i} \\ &\quad \times \prod_{j=1}^{n_2}\frac{\beta+j}{\alpha+\beta+n_1+j}.\end{aligned}$$

Now that I have shown how to estimate the transition probabilities given a sequence of 0's and 1's, I will outline a method to estimate the de Bruijn word length which was most likely used to generate the sequence. This is a much harder problem when linked with the transition probabilities, since we require a different number of transition probabilities for different word lengths, $m$. Since the word lengths can only take integer values, I have chosen to proceed using a method of model comparison called Bayes' factors (Kass and Raftery, 1995; O'Hagan, 1997). This is based on the Bayesian approach to hypothesis testing where we decide whether some given data has arisen under one of two different hypotheses. In terms of the de Bruijn process, we would question whether a sequence, $S$, of 0's and 1's was created from a word length $m_1$ de Bruijn process (hypothesis 1) with probability $P(S|m_1)$, or from a length $m_2$ de Bruijn process (hypothesis 2) with probability $P(S|m_2)$. For both of these, we would also have the prior probabilities, $P(m_1)$ and $P(m_2)$ respectively

that the sequence was indeed generated using a length $m_1$ or $m_2$ de Bruijn process. When combined with the data this then gives appropriate posterior probabilities, $P(m_1|S)$ and $P(m_2|S) = 1 - P(m_1|S)$. If we consider Bayes' theorem in terms of an odds scale of these hypotheses when in favour of $m_1$, then we come to the following equation:

$$\frac{P(m_1|S)}{P(m_2|S)} = \frac{P(S|m_1)}{P(S|m_2)} \frac{P(m_1)}{P(m_2)}.$$

If we say that the hypotheses, $m_1$ and $m_2$, are equally likely then we can define the Bayes' factor to be the posterior odds in favour of $m_1$:

$$B_{1,2} = \frac{P(S|m_1)}{P(S|m_2)}.$$

Since the transition probabilities are unknown parameters in this case, we find an expression for $P(S|m_k)$ by integrating over the parameter space. This becomes:

$$P(S|m_k) = \int \mathcal{L}(S|p_k, m_k) P(p_k|m_k) \ dp_k,$$

for $k \in \{1, 2\}$, where $\mathcal{L}(S|p_k, m_k)$ is the likelihood of the data and $P(p_k|m_k)$ is the prior density of the model parameters, $p$. We immediately notice the similarity between this and the model evidence in Equation (4.5) (for $m = 2$) and Theorem 4.21. Due to the fact that we have a conjugate relationship, we can now state that the Bayes' factor ratio is equivalent to the ratio of the model evidences for each of the model hypotheses, which is shown in Theorem 4.22. I note here that it is not necessary to calculate the posterior on the transition probabilities since the expression for the Bayes' factor is only dependent on the prior density. For each calculation of $P(S|m_k)$, we will know the length $m_k$ and hence the quantity of parameters, $p$, which are to be estimated.

In the set up of the de Bruijn process, I make the assumption that the word length, $m$, will remain fairly small. This is pragmatic as large word lengths create a vast number of transition probabilities to be estimated, and the increase in dimension does not have much effect on the accuracy of the estimates. Therefore, I will make the choice to limit word lengths to not be greater than 10. So, to choose the word

length that best represents the data, we calculate $B_{i,j} = \frac{P(S|m_i)}{P(S|m_j)}$ for each pair of models where $i, j \in \{1, 2, ..., 10\}$ and select the value for $m$ in which the Bayes' factor is consistently higher. When values of $B_{i,j}$ are large, this gives us more evidence that we should reject the model with word length $m_i$ in favour of the model with word length $m_j$.

By only selecting 10 different models to compare and choosing the one that best represents the data, this adds a frequentist aspect to our method. Instead, we could opt to do this in a fully Bayesian way to maximise the Bayes' factor and allow any word length to be considered. However, to do this we would have to put a fairly strong prior on $m$ to minimise large potential word lengths. This is left for future work.

**Theorem 4.22** (Estimation of De Bruijn Word length by Bayes' factors, $m \geq 1$)**.** Consider a sequence of 0's and 1's which was generated under one of two hypotheses. The first is a de Bruijn process with word length $m_1$ and the second is a de Bruijn process with word length $m_2$. The Bayes' factor ratio is as follows:

$$B_{1,2} = \frac{P(S|m_1)}{P(S|m_2)}$$

where,

$$\begin{aligned} P(S|m_k) &= \int P(S|p, m_k)P(p|m_k)dp \\ &= \prod_{i=0}^{2^{m_k}-1} \frac{\Gamma(n_i^{((2i+1) \bmod 2^{m_k})-1} + \beta_{i+1})\Gamma(n_i^{((2i+1) \bmod 2^{m_k})} + \alpha_{i+1})}{\Gamma(n_i^{((2i+1) \bmod 2^{m_k})-1} + n_i^{((2i+1) \bmod 2^{m_k})} + \beta_{i+1} + \alpha_{i+1})}, \end{aligned}$$

for $k \in \{1, 2\}$. When values of $B_{1,2}$ are large, we have more evidence to reject the first hypothesis with word length $m_1$ in favour of the second hypothesis with word length $m_2$.

*Proof.* Follows from Theorem 4.21. □

I would lastly like to point out that, when calculating $P(S|m_k)$ for each possible word length, we integrate over the unknown transition probabilities. This then removes this parameter from the equation and we are left with a result that is

independent of the transitions. The only knowledge that we need to estimate the word length is the prior distribution on the transition probabilities, and the number of times each transition takes place in the sequence. This is a highly efficient result since the number of transition probabilities changes dependent on the word length. Since we have a conjugate relationship, and are independent of the transitions, we find that it is then very computationally efficient to calculate $P(S|m_k)$ for each word length $m_k$ for each $k = 1, ..., 10$. Thus, given a sequence, once we have successfully estimated the word length $m$, we can then use the result given in Theorem 4.21 to estimate the corresponding transition probabilities for the chosen model.

### 4.3.1 Examples

I will now present three simple examples to illustrate how to perform the inference from Theorems 4.22 and 4.21 on a given sequence. This will include how to use Bayes' factors to select the most likely word length, $m$, and how to estimate the transition probabilities. The first given sequence, $S$, is shown in Figure 4.1. This sequence was created with a length $m = 3$ de Bruijn process with transition probabilities, $\{p_{000}^{001}, p_{001}^{011}, p_{010}^{101}, p_{011}^{111}, p_{100}^{001}, p_{101}^{011}, p_{110}^{101}, p_{111}^{111}\} = \{0.10, 0.80, 0.30, 0.85, 0.15, 0.70, 0.20, 0.90\}$. It has 500 time steps and is shown to be very sticky to both 0's and 1's (since the marginals are $\pi(0) = \pi(1) = 0.5$).

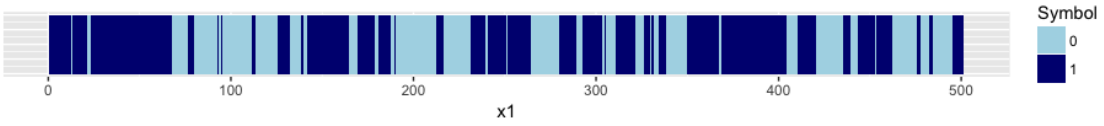



Fig. 4.1 Sample from a length $m = 3$ de Bruijn process with letters 0 and 1. The transition probabilities are: $\{p_{000}^{001}, p_{001}^{011}, p_{010}^{101}, p_{011}^{111}, p_{100}^{001}, p_{101}^{011}, p_{110}^{101}, p_{111}^{111}\} = \{0.10, 0.80, 0.30, 0.85, 0.15, 0.70, 0.20, 0.90\}$.

We begin the example by trying to estimate the word length using Bayes' factors with the model evidence stated in Theorem 4.22. The model evidence for the sequence, $S$, is calculated for each of the models, $m_k$. Each of these models represents the de Bruijn process for different word lengths, $m = 1, ..., 10$, for comparison. We require the prior distribution for the transition probabilities to calculate the model evidence. We do not assume any prior knowledge, so let each $\alpha = \beta = 1$ for the equivalence of

a Uniform prior. A Bayes' factor ratio is calculated for each pair of word lengths such that:

$$B_{i,j} = \frac{P(S|m_i)}{P(S|m_j)}, \qquad \text{for} \quad i, j = 1, ..., 10.$$

The ratios, $B_{i,j}$, produced for word lengths $m = 2, .., 8$ are shown in Table 4.4. The values given are on a logarithmic scale and I note that since the table is anti-symmetric about the leading diagonal, we only need to analyse the lower triangular section. When $B_{i,j} > B_{i',j}$ for all $i' = 2, ..., 8$ and $j = 2, ..., 8$, this indicates that the model, $m_i$, is preferred and that the sequence was created with a length $m = i$ word de Bruijn process. Looking at the table, we can see that the values in the column for $m = 3$ are always largest across each row. Therefore, we conclude that the sequence was generated using a length three de Bruijn process.

| m | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 2 | 0.00 | 5.91 | -0.81 | -13.85 | -34.14 | -55.76 | -75.15 |
| 3 | -5.91 | 0.00 | -6.72 | -19.75 | -40.05 | -61.66 | -81.05 |
| 4 | 0.81 | 6.72 | 0.00 | -13.03 | -33.33 | -54.94 | -74.33 |
| 5 | 13.85 | 19.75 | 13.03 | 0.00 | -20.30 | -41.91 | -61.30 |
| 6 | 34.14 | 40.05 | 33.33 | 20.30 | 0.00 | -21.62 | -41.00 |
| 7 | 55.76 | 61.66 | 54.94 | 41.91 | 21.62 | 0.00 | -19.39 |
| 8 | 75.15 | 81.05 | 74.33 | 61.30 | 41.00 | 19.39 | 0.00 |

Table 4.4 Table giving the log of Bayes' factors for 7 models with word lengths, $m = 2, ..., 8$ for the sequence shown in Figure 4.1.

I then estimated the word lengths of 1000 different sequences each of length 500 that were generated with a length $m = 3$ de Bruijn process with transition probabilities: $\{p^{001}_{000}, p^{011}_{001}, p^{101}_{010}, p^{111}_{011}, p^{001}_{100}, p^{011}_{101}, p^{101}_{110}, p^{111}_{111}\} = \{0.10, 0.80, 0.30, 0.85, 0.15, 0.70, 0.20, 0.90\}$. The distribution of these estimated word lengths is shown in the histogram in Figure 4.2. We can see here that although the majority of the sequences are estimated to have a word length of three, it is possible to produce sequences from the same transition probabilities that are actually more similar to sequences produced with word lengths one, two or four. This is to be expected and if we increased the length of the sequences, the proportion of sequences estimated to have the correct word length would also increase.

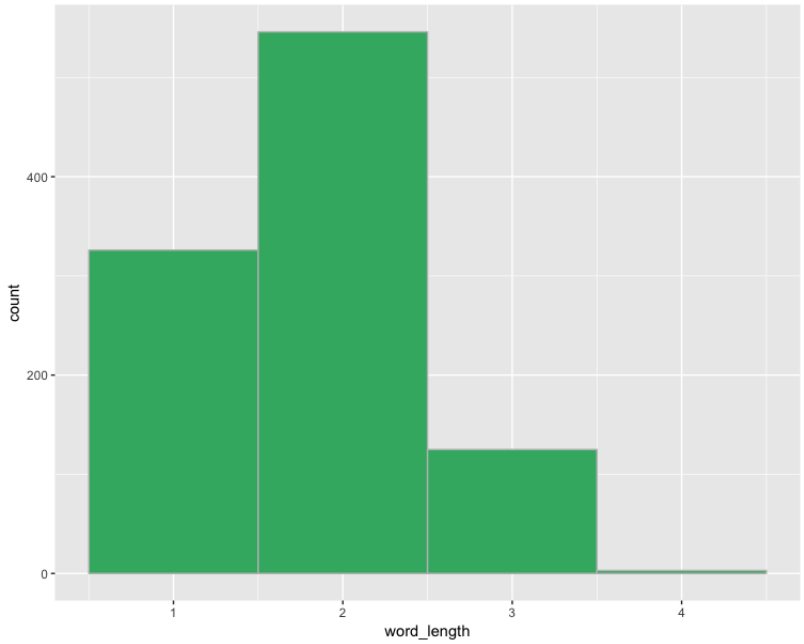



Fig. 4.2 Histogram of estimated word lengths from the example in Figure 4.1.

Having estimated the word length to be $m = 3$, the corresponding transition probabilities for the sequence are now estimated. This can be done by either using a maximum likelihood approach or by using a simple Metropolis Hastings MCMC. By taking the Bayesian approach using the likelihood in Theorem 4.20, and assuming we have uninformative priors, I estimated the parameters to be: $\{0.108, 0.809, 0.278, 0.852, 0.168, 0.658, 0.177, 0.902\}$. Comparing this with the de Bruijn process that was used to generate the sequence shows that the estimation has proven to be successful with minimal error uncertainties on estimates.

The sequence for a second example is shown in Figure 4.3. This example differs from the first one above since the marginal probabilities for the letters are not equal. Instead I have set $\pi(\{0\}) = 0.2$ and $\pi(\{1\}) = 0.8$. The sequence is of length 500 and was generated with a length $m = 2$ de Bruijn process with transition probabilities: $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\} = \{0.775, 0.7, 0.825, 0.9\}$. Therefore, we still expect the 1's to be highly sticky.

The word length for the given sequence is estimated using the same method as explained for the previous example. This includes calculating the model evidence used for the Bayes' factor for de Bruijn processes with word lengths $m = 1, ..., 10$, and then calculating the Bayes' factor ratio for every possible pair of models. Table

4.3 shows the results from this and shows that the most likely word length to have generated the sequence is in fact $m = 2$ since all values in the second column are always the largest across each row. Again, these values are converted to the logarithmic scale and we need only focus on the lower triangular section of the table due to the anti-symmetry about the leading diagonal. Given the word length was estimated to be two, we can then estimate the corresponding transition probabilities. Using a simple Metropolis Hastings MCMC, I obtained the following estimates: $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\} = \{0.768, 0.688, 0.819, 0.897\}$.

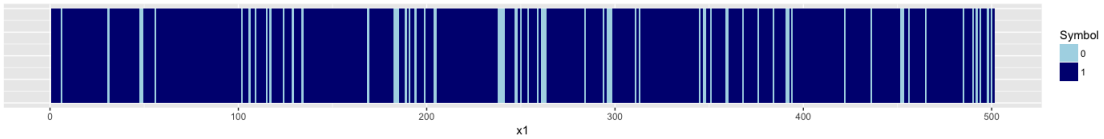



Fig. 4.3 Sample from a length $m = 2$ de Bruijn process with letters 0 and 1. The transition probabilities are: $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\} = \{0.775, 0.7, 0.825, 0.9\}$.

| m | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0.00 | 14.17 | 3.12 | -5.99 | -17.90 | -36.45 | -51.65 |
| 2 | -14.17 | 0.00 | -11.06 | -20.17 | -32.07 | -50.62 | -65.83 |
| 3 | -3.11 | 11.06 | 0.00 | -9.10 | -21.01 | -39.56 | -54.76 |
| 4 | 5.99 | 20.17 | 9.10 | 0.00 | -11.90 | -30.45 | -45.66 |
| 5 | 17.90 | 32.07 | 21.01 | 11.90 | 0.00 | -18.55 | -33.76 |
| 6 | 36.45 | 50.62 | 39.56 | 30.45 | 18.55 | 0.00 | -15.21 |
| 7 | 51.65 | 65.83 | 54.76 | 45.66 | 33.76 | 15.21 | 0.00 |

Table 4.5 Table giving the log Bayes' factors for 7 models with word lengths, $m = 1, ..., 7$ for the sequence shown in Figure 4.3.

I again then estimated the word lengths of 1000 different sequences each of length 500 that were generated with a length $m = 2$ de Bruijn process with transition probabilities: $\{p_{00}^{01}, p_{01}^{11}, p_{10}^{01}, p_{11}^{11}\} = \{0.775, 0.7, 0.825, 0.9\}$. The distribution of these are shown in the histogram in Figure 4.2. Comparing this with the histogram in Figure 4.2, we can see that the number of sequences with the correct estimated word length ($m = 2$) has greatly increased with very few shown to be slightly lower or higher. This may be because all of the transition probabilities in this example have a significant effect on the resulting sequence and so we are sure of the word length every time.

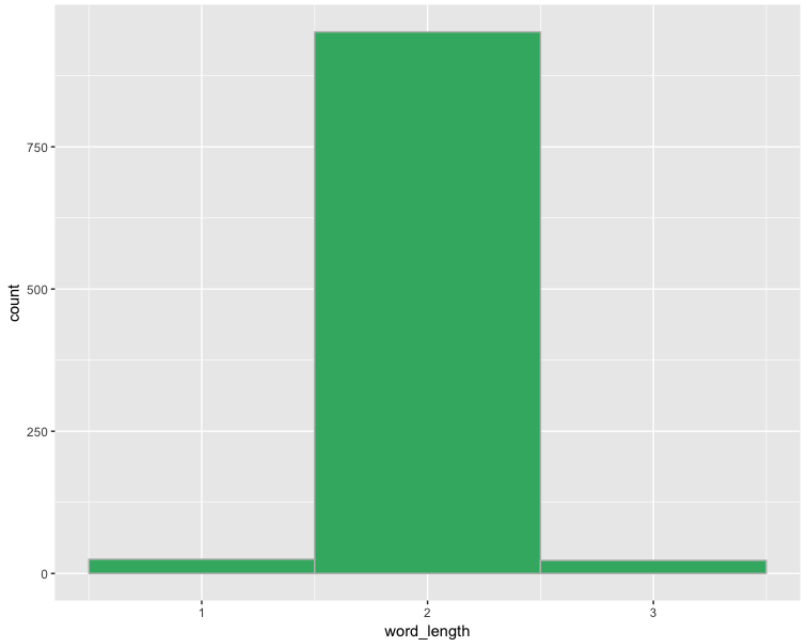



Fig. 4.4 Histogram of estimated word lengths from the example in Figure 4.3.

The sequence for the final example is shown in Figure 4.5. This is the same example as from Figure 3.9 where I showed that some de Bruijn processes can collapse down to other de Bruijn processes with shorter word lengths. The example shows a sequence of length 500 generated from a length $m = 4$ de Bruijn process where the marginal probabilities are equal ($\pi(\{0\}) = \pi(\{1\}) = 0.5$) and the transition probabilities are such that there is a 10% chance of swapping letter and a 90% chance of remaining at the same letter.

After calculating the Bayes' factors in the same way as the previous two examples, the results are shown in Table 4.5. As expected, the results imply that the sequence was generated using a length $m = 1$ de Bruijn graph, which is equivalent to just a classical Markov chain. It gives a lower estimate because the sequence of letters is indistinguishable from a sequence produced with a shorter word length. The intermediate transition probabilities are not adding any further structure in the model and are hence unnecessary. Given the estimate of $m = 1$, the transition probabilities for the sequence are estimated to be $p_0^1 = 0.102$ and $p_1^1 = 0.902$. Hence we can conclude that the sequence could have equally been generated with a length $m = 1$ de Bruijn process with transition probabilities $p_0^1 = 0.1$ and $p_1^1 = 0.9$.

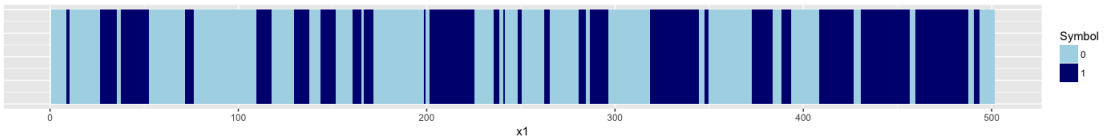



Fig. 4.5 Sample from a length $m = 4$ de Bruijn process with letters 0 and 1. The transition probabilities are: $\{p^{0001}_{0000}, p^{0011}_{0001}, p^{0101}_{0010}, p^{0111}_{0011}, p^{1001}_{0100}, p^{1011}_{0101}, p^{1101}_{0110}, p^{1111}_{0111}, p^{0001}_{1000}, p^{0011}_{1001}, p^{0101}_{1010}, p^{0111}_{1011}, p^{1001}_{1100}, p^{1011}_{1101}, p^{1101}_{1110}, p^{1111}_{1111}\} = \{0.1, 0.9, 0.1, 0.9,\ 0.1, 0.9, 0.1, 0.9, 0.1, 0.9, 0.1, 0.9, 0.1, 0.9, 0.1, 0.9\}$.

| m | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0.00 | -7.24 | -15.72 | -24.49 | -36.85 | -56.61 | -68.84 |
| 2 | 7.24 | 0.00 | -8.48 | -17.25 | -29.61 | -49.37 | -61.61 |
| 3 | 15.72 | 8.48 | 0.00 | -8.77 | -21.13 | -40.89 | -53.12 |
| 4 | 24.49 | 17.25 | 8.77 | 0.00 | -12.36 | -32.12 | -44.35 |
| 5 | 36.85 | 29.61 | 21.13 | 12.36 | 0.00 | -19.76 | -31.99 |
| 6 | 56.61 | 49.37 | 40.89 | 32.12 | 19.76 | 0.00 | -12.23 |
| 7 | 68.84 | 61.61 | 53.12 | 44.35 | 31.99 | 12.23 | 0.00 |

Table 4.6 Table giving the log Bayes' factors for 7 models with word lengths, $m = 1, ..., 7$ for the sequence shown in Figure 4.5.

This is emphasised by the histogram in Figure 4.6 which shows the distribution of the estimated word lengths of 1000 different sequences each of length 500. Each of these sequences was created using the $m = 4$ length de Bruijn process (Figure 4.5) but for the majority of them, the Bayes' factor method suggests that an $m = 1$ length de Bruijn process was more likely used.

## 4.4 Discussion

In this chapter, I have extended the ideas from Chapter 3 to look at both properties of de Bruijn processes and inference. The main property that I have focused on is the run length of 1's that appear in the sequences of 0's and 1's generated from the de Bruijn processes. I have presented expressions for the run length distribution, expectation, variance and generating functions for both $m = 2$ and for general word lengths greater than three. Due to similarities with the geometric distribution, I was able to use known results to help simplify expressions where possible. I have included examples to compare the simulated run lengths from Figure 3.4 in Chapter 3 to the theoretical expected run lengths that I have calculated here. They have

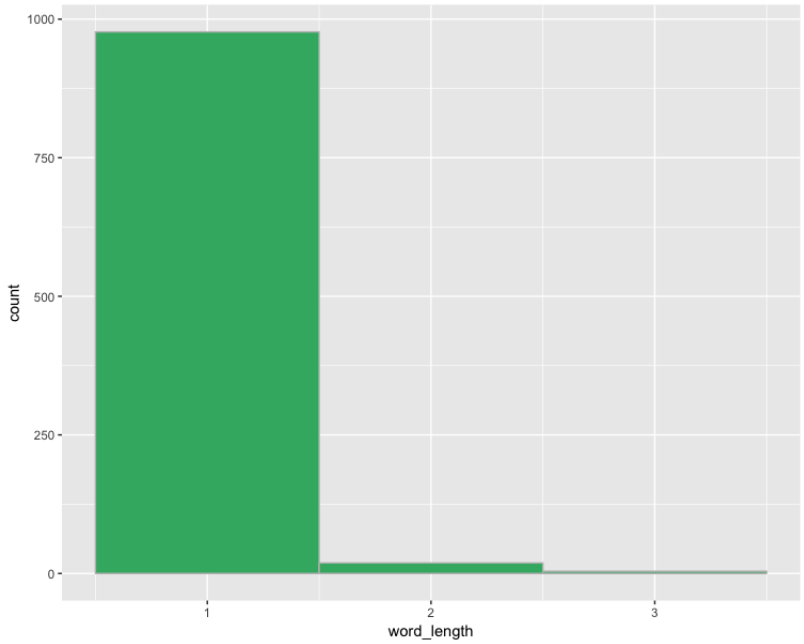



Fig. 4.6 Histogram of estimated word lengths from the example in Figure 4.5.

shown to mostly agree, with the exception that we see more variance in the simulated expected run lengths.

I then went on to look at the inference for de Bruijn processes and have shown how to use a simple Bayes' factor method to choose the word length that best fits a given sequence. Once we have estimated the word length, it is then possible to estimate the transition probabilities using either maximum likelihood or using Bayesian MCMC.

In the next Chapter I will be discussing ways to extend the de Bruijn process to two dimensions (and higher). This is a difficult problem since there is not a natural direction in a two dimension grid, that is a key feature of de Bruijn graphs.

# Chapter 5

# De Bruijn Processes in Two Dimensions (and Higher)

## 5.1 Introduction

In Chapters 3 and 4, I demonstrated that there was a need for a correlated Bernoulli process, which would offer an improvement to current classification methods, such as logistic regression (Chang et al., 2016; Diggle et al., 1998). To do this, I defined a de Bruijn process, including methods of simulation and inference. A run length distribution was also derived for the de Bruijn process which led to calculation of an expected run length, variance of run length and generating functions. Several examples were given to show the effectiveness of the method.

The natural next stage of this work, which I will develop in this chapter, is a two dimensional correlated Bernoulli process. Although it would be simpler to progress in the same way as the 1-d version, the distinct directionality in the de Bruijn structure cannot easily be applied to a 2-d grid. Therefore, we must consider whether it is better to adapt the 1-d directional methodology, including direction in 2-d in some way, or if it is better to develop a new formulation for the correlated Bernoulli process.

In this chapter I will attempt the former, creating a directional multi-dimensional correlated Bernoulli process using de Bruijn graphs. In the following sections, I outline two different methods for generating such a process. The first method (Method 1)

takes advantage of the structures of multivariate and higher-order Markov chains (Ching and Ng, 2006; Ching et al., 2008), whilst the second method (Method 2) focuses on defining the 2-d equivalent of a de Bruijn word. The second method proves to be the more successful of the two, requiring fewer parameters, and allows us to perform inference. The inference presented is equivalent to the inference method in Chapter 4, as we are able to calculate the 1-d equivalence for each of the 2-d de Bruijn words generated. Hence, each 2-d de Bruijn process is shown to be treatable in the same way as a 1-d problem. Examples are given, along with a motivation for a 2-d version of the run length distribution. Although not discussed at length, insights on how to extend Method 2 to higher dimensions are also considered.

## 5.2 Towards a 2-d De Bruijn Process

To begin, we will discuss Markov random fields (Rue and Held, 2005; Winkler, 1995), since they are a natural extension of a Markov chain to higher dimensions.

A random field, $x$, is a random function defined by its joint distributions. For the random field to be Markov, it must satisfy the following constraint:

$$P(x_i|\{x_j : j \neq i\}) = P(x_i|\{x_j : j \in S_i\}),$$

where $S_i$ is a neighbourhood of points, $\{s_j : j \in S_i\}$, to the point, $s_i$. Hence, the distribution of a point within a Markov random field is only conditional on the points within a specified neighbourhood and nothing else. In practice, Gauss Markov random fields are used when $x$ is Normally distributed with mean and covariance functions. For such Markov random fields to be applicable for a 2-d correlated Bernoulli process, we require that the random fields are discrete in space, and so we may want to consider a binary Markov random field (Possolo, 1986; Smith and Smith, 2006).

One particular type of Markov random field is a conditional autoregressive (CAR) model (Banerjee et al., 2004; Besag, 1974), which lives on lattice grids or regions. Each variable $x_{i,j}$ is conditional on the neighbourhood of points, such that the

probability distribution of $x_{i,j}$ is independent of non-neighbouring points. Often, a neighbourhood consists of the four surrounding points, $x_{i-1,j}$, $x_{i+1,j}$, $x_{i,j-1}$ and $x_{i,j+1}$, but it is possible for this to be extended. The outputs of a CAR model are calculated by specifying the set of full conditional distributions satisfying a form of auto-regression. These conditional distributions are often Normal, but can also be taken from the exponential family.

Therefore, a Bernoulli CAR model may be appropriate for the 2-d correlated Bernoulli process, and I may be able to use the concept of de Bruijn words to help define different possible neighbourhoods depending on the amounts of correlation required.

A similar concept to a Markov random field is a Markov mesh, introduced in Abend et al. (1965). There, the authors extend a Markov chain to a 2-d Markov mesh which follows a similar structure to a Markov random field, where each point is dependent on its nearest neighbours. They also discuss whether the neighbourhood dependence of each point need only be on those points in the upper left sector, rather than all surrounding points. Hence, this incorporates a notion of direction.

Another possible approach, which has a similar structure to Markov chains and de Bruijn graphs, is cellular automata (Agapie et al., 2014, 2004; Wolfram, 2002). Cellular automata (CA) define a discrete model consisting of a set of grid cells, where each of these cells takes the value of one of a set of pre-defined states. Often these states are just 'on' and 'off' or 'black' and 'white', and so there is an obvious connection with the type of grids being considered here. The CA model is of particular interest as the value of a particular cell depends on a set of rules based on the values of neighbouring cells. These rules are applied iteratively to each cell until a desired outcome is reached. There are many alterations to the basic model, including probabilistic cellular automata and Conway's game of life (1970).

CA models initially appear to be ideal for generating 2-d correlated Bernoulli simulations, as they are able to produce a grid of cells that each take one of two different states, 0 or 1. We would then need to generate a set of rules similar to the transition probabilities of Chapter 3 and 4, allowing us to iteratively produce the final

grid pattern. In particular, we may be interested in probabilistic cellular automata, discussed in Agapie et al. (2014), as we require a structure that is analogous to the de Bruijn transition probabilities. Agapie et al. (2014) consider a model with an $N$-length binary string consisting of the values 0 and 1. At each time step, one cell is allowed to flip, and each cell has probability $\frac{1}{N}$ of being selected to flip. If selected, then the probability that the cell will flip is dependent on how many 1's there are in the neighbourhood of that cell. The Markov chain associated with this process comprises of the $2^N$ possible binary strings of length $N$, where each string can transform to any other string that is at most one cell flip in difference. The stationary distribution is also provided in their article. Although I believe their work could be extended to 2 dimensions, since there is no focus on direction, the set up seems to have similarities with the multivariate Bernoulli distribution discussed by Teugels (1990). Hence, it may have similar issues such as the high number of parameters

Both cellular automata and probabilistic cellular automata simulate grids of 0's and 1's iteratively. One of the main advantages of simulating iteratively is that it removes any of the directionality that was present in the 1-d de Bruijn process, similar to the work by Besag (1986) on removing noise from pixelled images. Although this sounds promising, it is a large step away from the de Bruijn process described in Chapters 3 and 4, and so I leave non-directional methods for discussion in Section 6.

Finally, we notice that cellular automata lack the de Bruijn structure that determines how wide the dependent neighbourhood is for each cell. The model structure may need to be changed so that it is also dependent on how many cells are included in each neighbourhood. This could be treated in a similar way to the de Bruijn word length. The majority of work on both probabilistic and non-probabilistic cellular automata has been published by the same authors, implying that there has either been little interest in the subject, or that its applications are limited. Due to this and the other reasons stated above, I do not proceed with this method any further, and instead attempt to find an approach that has more similarities with de Bruijn processes.

By trying to progress in a direction that is more closely related to de Bruijn processes, I discovered that the term '2-d de Bruijn graph' does in fact already exist in the literature. However, after further investigation, I found that such methods did not have the desired properties I am looking for. In fact, multi-dimensional de Bruijn graphs normally refer to 1-d de Bruijn graphs where the word length is large. Thus, I began considering alternatives for two dimensional de Bruijn graphs.

The first thing to consider when trying to build a two dimensional de Bruijn graph is direction. As described previously, direction is a key aspect of a de Bruijn graph, but it is not clear what form the direction takes on a 2-d grid. Initially, to retain directionality, one can think of the space as being a 2-d grid, and consider having two separate de Bruijn graphs - one in the $x$ direction and one in the $y$ direction. Starting from the bottom left hand corner, we may then be able to simulate a grid from a pre-defined initial condition. Although this is unlikely to cause problems when considering its distributions and expectations, we have to ensure that there are no contradictions in the simulation. The contradictions occur because it is possible to simulate each point from two different directions ($x$ and $y$). Hence, for consistency, we would have to make sure that the probability of getting a 1 when simulating in both directions at the same point was the same.

The second problem with such a model is that it ignores any correlation in the diagonals. If you move only along the horizontal and vertical lines, then you are ignoring the diagonal dependencies, which are clearly important for the full correlation structure. Therefore, it may be more appropriate to build a de Bruijn graph that simulates along the two dimensions simultaneously. I develop two possible methods for this, which are described in the following sections. The first of these is based on averaging over several 1-d de Bruijn graphs, whilst the second considers what form a de Bruijn word may take in two dimensions.

### 5.2.1 Method 1

The connection between de Bruijn graphs and Markov chains means that we can consider building a two dimensional Bernoulli process using multivariate Markov

chains. Multivariate Markov chains are introduced by Ching and Ng (2006) and Ching et al. (2008) to model multiple categorical data sequences that have been generated by a similar source or have notable correlation between them. They consider $s$ different categorical sequences, each with $v$ possible states in the set $V = \{1, 2, ..., v\}$. Each $j^{\text{th}}$ sequence has a state vector, $\mathbf{X}_t^{(j)}$, at time $t$ where, $\mathbf{X}_t^{(j)} = \mathbf{e}_l = (0, ..., 0, 1, 0, ..., )$, if the $j^{\text{th}}$ sequence is in state $l$ at time $t$. The state probability distribution of the $j^{\text{th}}$ sequence at time $t+1$ depends on the state probabilities of all $s$ sequences at time $t$ and so they propose a multivariate Markov chain model as follows:

$$\mathbf{x}_{t+1}^{(j)} = \sum_{k=1}^{s} \lambda_{jk} \mathbf{P}^{(jk)} \mathbf{x}_t^{(k)}, \qquad \text{for} \quad j = 1, 2, ..., s \quad \text{and} \quad t = 0, 1, ...$$

where the weights, $\lambda_{jk}$, are non-negative real numbers such that,

$$\lambda_{jk} \geq 0, \quad 1 \leq j, \quad k \leq s \quad \text{and} \quad \sum_{k=1}^{s} \lambda_{jk} = 1, \quad \text{for} \quad j = 1, 2, ..., s \ .$$

$\mathbf{P}^{(jk)}$ is the one-step transition probability matrix from the states at time $t$ in the $k^{\text{th}}$ sequence to the states at time $t+1$ in the $j^{\text{th}}$ sequence. $\mathbf{x}_t^{(k)}$ is the state probability distribution of the $k^{\text{th}}$ sequence at time $t$, where, $\mathbf{x}_0^{(j)}$, is the initial probability distribution of the $j^{\text{th}}$ sequence. The state probability distribution of the $k^{\text{th}}$ sequence at time $(r+1)$ depends on the weighted average of $P^{(jk)}\mathbf{X}_r^{(k)}$. In matrix form, this becomes:

$$\mathbf{X}_{t+1} \equiv \begin{pmatrix} \mathbf{x}_{t+1}^{(1)} \\ \mathbf{x}_{t+1}^{(2)} \\ \vdots \\ \mathbf{x}_{t+1}^{(s)} \end{pmatrix} = \begin{pmatrix} \lambda_{11}P^{(11)} & \lambda_{12}P^{(12)} & \cdots & \lambda_{1s}P^{(1s)} \\ \lambda_{21}P^{(11)} & \lambda_{22}P^{(22)} & \cdots & \lambda_{2s}P^{(2s)} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{s1}P^{(s1)} & \lambda_{s2}P^{(s2)} & \cdots & \lambda_{ss}P^{(ss)} \end{pmatrix} \begin{pmatrix} \mathbf{x}_t^{(1)} \\ \mathbf{x}_t^{(2)} \\ \vdots \\ \mathbf{x}_t^{(s)} \end{pmatrix}.$$

We can therefore consider multivariate Markov chains as a way to connect many different 1-d Markov chains so that there is correlation spread across the ensemble. The weights, $\lambda$, act as a correlation parameter, controlling how much influence each individual Markov chain has in the whole multivariate chain.

Let's consider an example with two sequences, $S_1$ and $S_2$ which are known to have a degree of correlation between them. This gives us the multivariate Markov chain as follows for each of the sequences, $S_1$ and $S_2$:

$$\begin{aligned}
\mathbf{x}_{t+1}^{(S_1)} &= \lambda_{S_1S_1}\mathbf{P}^{(S_1S_1)}\mathbf{x}_t^{(S_1)} + \lambda_{S_1S_2}\mathbf{P}^{(S_1S_2)}\mathbf{x}_t^{(S_2)}, \\
\mathbf{x}_{t+1}^{(S_2)} &= \lambda_{S_2S_1}\mathbf{P}^{(S_2S_1)}\mathbf{x}_t^{(S_1)} + \lambda_{S_2S_2}\mathbf{P}^{(S_2S_2)}\mathbf{x}_t^{(S_2)}, \qquad \text{for} \quad t = 0, 1, ...
\end{aligned}$$

where in matrix form, this is as follows:

$$\mathbf{X}_{t+1} \equiv \begin{pmatrix} \mathbf{x}_{t+1}^{(S_1)} \\ \mathbf{x}_{t+1}^{(S_2)} \end{pmatrix} = \begin{pmatrix} \lambda_{S_1S_1}P^{(S_1S_1)} & \lambda_{S_1S_2}P^{(S_1S_2)} \\ \lambda_{S_2S_1}P^{(S_2S_1)} & \lambda_{S_2S_2}P^{(S_2S_2)} \end{pmatrix} \begin{pmatrix} \mathbf{x}_t^{(1)} \\ \mathbf{x}_t^{(2)} \end{pmatrix}.$$

For the model above, we require four values for $\lambda$ and four different transition matrices, $\mathbf{P}$. Each of the transition matrices, $\mathbf{P}^{(S_iS_j)}$, give the probabilities of transitioning from all of the states in $S_i$ to all of the states in $S_j$. For example, say both $S_1$ and $S_2$ have possible states $V = \{v_1, v_2, v_3\}$. Then $P^{(S_1S_2)}$ is a $3 \times 3$ size matrix, where the elements in $P^{(S_1S_2)}$ each give the probability of transitioning from the state $v_i$ in $S_1$ to the state $v_j$ in $S_2$ for all $i, j = 1, 2, 3$.

It is important to note here that the authors (Ching and Ng (2006) and Ching et al. (2008)) also introduced a higher order Markov chain (as mentioned in Section 3.2), for which there are many similarities with de Bruijn graphs. Higher order Markov chains have a fairly similar structure to multivariate Markov chains since the model becomes the weighted average of transition probabilities from the previous $n$ states. Abend et al. (1965) utilise higher-order Markov chains for the classification of binary random patterns, where they consider black and white images to be two dimensional arrays of binary random variables. Although they appear to be identical in the output produced to a de Bruijn graph, I believe that de Bruijn graphs offer a different layout that is simpler to follow. The total number of parameters for the higher-order Markov chain is of order $ns^2$ (where $n$ is the order of the Markov chain and $s$ is the number of states), but the total number of parameters for the similar de Bruijn graph is of order $2^m$ (where $m$ is the de Bruijn word length).

Although the authors do also consider a higher-order multivariate Markov chain model, due to my previous work and simplicity in defining these sequences as de Bruijn graphs, I have chosen to just adapt the first-order multivariate Markov chain model to fit my layout in de Bruijn graphs. This is possible because de Bruijn graphs are Markov on the 'word'.

In the work of Ching and Ng (2006), correlation is included across $s$ different sequences that are known to be generated from a similar source. For it to be possible to use the ideas from this method to help create a two dimensional correlated Bernoulli process, we must define a model input space that lies on a 2-d grid. We thus define each row of symbols (or letters) in the model space to be the $s$ individual sequences with known correlation in the multivariate Markov chain framework. Therefore, we are able to simulate grids of letters by initiating the Markov chain from one edge and moving across, as we would in an ordinary 1-d Markov chain.

To fit within the de Bruijn framework, each state in the multivariate Markov chain is a word of 0's or 1's that have length, $m$, so that like the 1-d case, words are dependent on previous words in the chain. The notion of transitioning from word to word in a simulation is lost slightly, since we focus on whether a specific point is a 0 or a 1, dependent on the words that come before. The word length, as defined previously for the 1-d de Bruijn graph, requires multivariate Markov chains to be adapted further. In the general multivariate Markov model, the correlation between sequences is spread over all rows. In this case, we define how much to spread correlation via the word length, and so reduce how many rows are included by only selecting those that are within the de Bruijn word neighbourhood. In 1-d de Bruijn graphs, each point is dependent on the previous word which consists of the $m$ points that come before. Similarly, in the 2-d case, each point is now specified to be dependent on the points in the $m$ closest rows above and below in the input space. In terms of de Bruijn words, we have each point being dependent on the previous $m$ length word along the same row, lower diagonal and upper diagonal. The model is written as follows, where the probability distribution of the $j^{\text{th}}$ sequence of 0's and 1's is dependent on the probabilities of the $m$ closest sequences in terms of the words

at time $t$:

$$\mathbf{x}_{t+1}^{(j)} = \sum_{k=j-1}^{j+1} \lambda_{jk} \mathbf{P}^{(jk)} \mathbf{x}_t^{(k)}, \qquad \text{for} \quad j = 1, 2, ..., s \quad \text{and} \quad t = 0, 1, ...$$

Since the transition matrix, $\mathbf{P}^{(jk)}$, gives the transitions in terms of words, and we are still working with something that is Markov on the word and not the letter, the summation only sums over three quantities. This is because the $m$ rows either side of sequence $j$ are contained within the two diagonal $m$ length de Bruijn words. Hence, regardless of the word length, it is only ever necessary to average over three probabilities.

The $m = 2$ structure is shown in Figure 5.1, where the green point is dependent on the current row, as well as the two rows above and below. This translates to the green point being dependent on the three words of length two in the three different directions. The blue points represent 0's and the orange points represent 1's. We can see that the green point is dependent on the $m = 2$ rows above and below the current row, but it is due to the de Bruijn word property that we need only consider the average probability in these three directions.

For a 2-d 3-word de Bruijn process, we would expand this to include the six closest sequences to the current sequence chosen (three above and three below). Each point would be dependent on the weighted average of the three length $m = 3$ words along the same row, the upper diagonal and the lower diagonal. This is equivalent to saying that each letter is dependent on the de Bruijn graphs running in three different directions; along the same row, the upper diagonal and the lower diagonal (shown in the diagram in figure 5.1).

The de Bruijn word length, $m$, determines how many rows are included in the correlation, and, since I am defining the states to be words of length $m$, this is equivalent to having three separate (but connected) de Bruijn graphs. If we then simulate the grid of letters from the left edge, a transition matrix is needed for each row, down diagonal and up diagonal in the input space. Following Equation 5.2.1, we are taking the weighted average over the transition probabilities for the three

de Bruijn graphs, which hence provides a direction from left to right as in the one dimensional case.

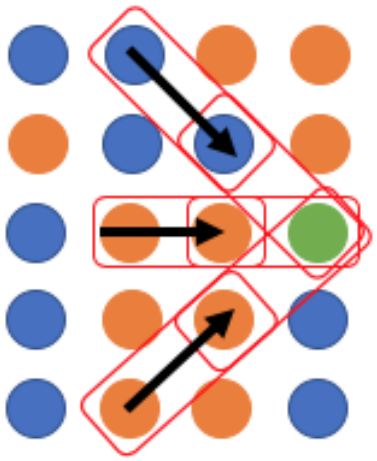


Fig. 5.1 2-d de Bruijn Graph example with word length $m = 2$ to show the three weighted de Bruijn graphs that the green symbol is dependent on. The blue points represent 0's and the orange points represent 1's.

So far, each weight, $\lambda$, is taken to be $\frac{1}{3}$, as I am not assuming the correlation in any one direction is stronger than in any other direction. This can be changed to add weight to a particular direction, or we may consider ways to estimate these weights for a given example. This is discussed in Ching et al. (2008) where they find $\lambda$ to minimise the maximum distance between $\sum_{k=1}^{m} \lambda_{jk} \hat{P}^{(jk)} \hat{\mathbf{x}}^{(k)}$ and $\hat{\mathbf{x}}^{(j)}$ subject to $\sum_{k=1}^{s} \lambda_{jk} = 1$ and $\lambda_{jk} \geq 0, \ \forall\, k$. Here, $\hat{\mathbf{x}}$, is the proportion of the occurrence of each state (or word) in each of the sequences.

Method 1 has to be adapted slightly when we are at the edges of the input space and the number of letters in the diagonal de Bruijn graphs is less than the word length, $m$. Here, we would have insufficient knowledge of all prior $m$ letters, and we instead use the maximum word length possible for these de Bruijn graphs. This means that the very edge letters have no dependencies in one direction, and so the contributed probability for the weighted equation above is drawn from an independent Bernoulli distribution. For the next row in, this would then be the equivalent to Markov since we know about one other letter. This continues for each row until we reach the word length required for the remaining de Bruijn graphs.

### Examples and Extensions to Higher Dimensions

Two examples of 2-d de Bruijn processes with word length $m = 3$ are shown in figure 5.2. Both processes are run long enough to reach a grid of size $8 \times 70$. The top plot is the simulated 2-d sequence when the de Bruijn word length is $m = 0$, and so each letter is simulated from Bernoulli trials with probability 0.5. The dark blue points represent simulated 1's and the light blue points represent simulated 0's. They are shown to be equally and randomly distributed, as expected. The bottom plot is constructed such that the 0's and 1's are forced to be very sticky, but the marginal probabilities remain at 0.5. The dark blue and light blue points now occur in patches, which is the 2-d equivalent to the large run lengths seen in the 1-d version.

To generate both of these plots 24 different transition matrices are required, to reach a total of 158 transition probabilities. There are 158 such probabilities required because a transition matrix for each row, upwards diagonal and downwards diagonal is required. In the bottom plot, for the rows and inner diagonals where we have not reached the edge, I have set the transition probabilities to be approximately $\{p^{001}_{000}, p^{011}_{001}, p^{101}_{010}, p^{111}_{011}, p^{001}_{100}, p^{011}_{101}, p^{101}_{110}, p^{111}_{111}\} = \{0.1, 0.7, 0.3, 0.8, 0.2, 0.7, 0.3, 0.9\}$ to create a sticky grid for both 0's and 1's. As we get closer to the edges of the top and bottom of the grid, we must reduce the length of the word for the corresponding diagonal de Bruijn graph. Here I have chosen the transitions $\{p^{01}_{00}, p^{11}_{01}, p^{01}_{10}, p^{11}_{11}\} = \{0.1, 0.8, 0.2, 0.9\}$ and $\{p^{1}_{0}, p^{1}_{1}\} = \{0.1, 0.9\}$. For simulating points on the far edges of the grid, we have no information for that corresponding edge, and so I just state that there is a 50% chance of a 1 for that diagonal de Bruijn graph. The letters in the top plot are generated using the equivalence of Bernoulli trials, and I thus set all 158 transition probabilities to be 0.5.

Figure 5.3 shows another example of a word length $m = 3$ 2-d de Bruijn process. As before, the transition probabilities are chosen such that the process is very sticky to both 0's and 1's. The top plot shows the simulation produced, where the dark blue points represent 1's and the light blue points represent 0's. The middle plot shows the probability of getting the letter 1 at each point in the grid, calculated after averaging the three separate de Bruijn graphs. We can see that generally, this

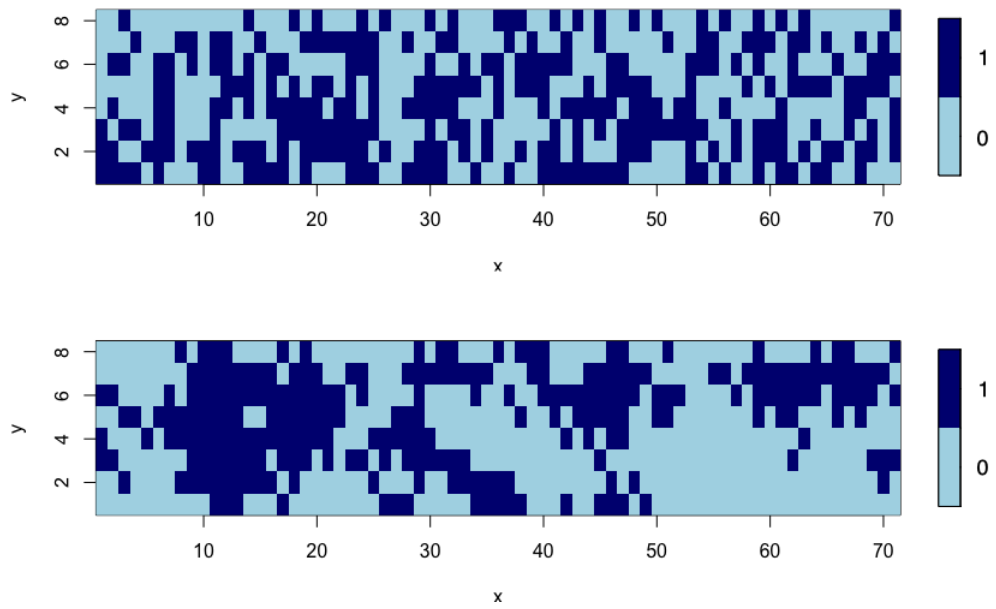



Fig. 5.2 Plots showing 2-d simulations of 0's and 1's generated from a Bernoulli distribution (top) and a word length 3 de Bruijn process (bottom). The transition probabilities for the bottom plot are such that the points are sticky to both 0's and 1's.

pattern closely reflects the final 0/1 classification, where we can see that the edges of the patches in the top plot have close to a 50% chance of being either a 0 or a 1. The bottom plot is similar, but instead shows the range of probabilities (difference between the smallest and largest probabilities) from the three de Bruijn graphs in each direction. Darker areas represent a larger range in probabilities. Although it is not very clear in this plot, we can see that the darker areas tend to occur near the boundary of each cluster of 0's or 1's.

We now consider whether this method is extendable to higher dimensions. If we first consider the step up from two dimensions to three, we can see that we will require an extra two diagonal de Bruijn graphs when averaging in each direction. The de Bruijn processes can still be simulated from left to right across a plane, but we will now have additional diagonal dependencies. Hence, at each point we have the average of 5 probabilities instead of 3. For even higher dimensions, we would continue to add another 2 diagonal de Bruijn graphs in the extra directions of the dimension. Therefore, for a problem in $x$ dimensions, we require $2^x$ different de Bruijn graphs, and will end up averaging $2^x$ probabilities for each point required.

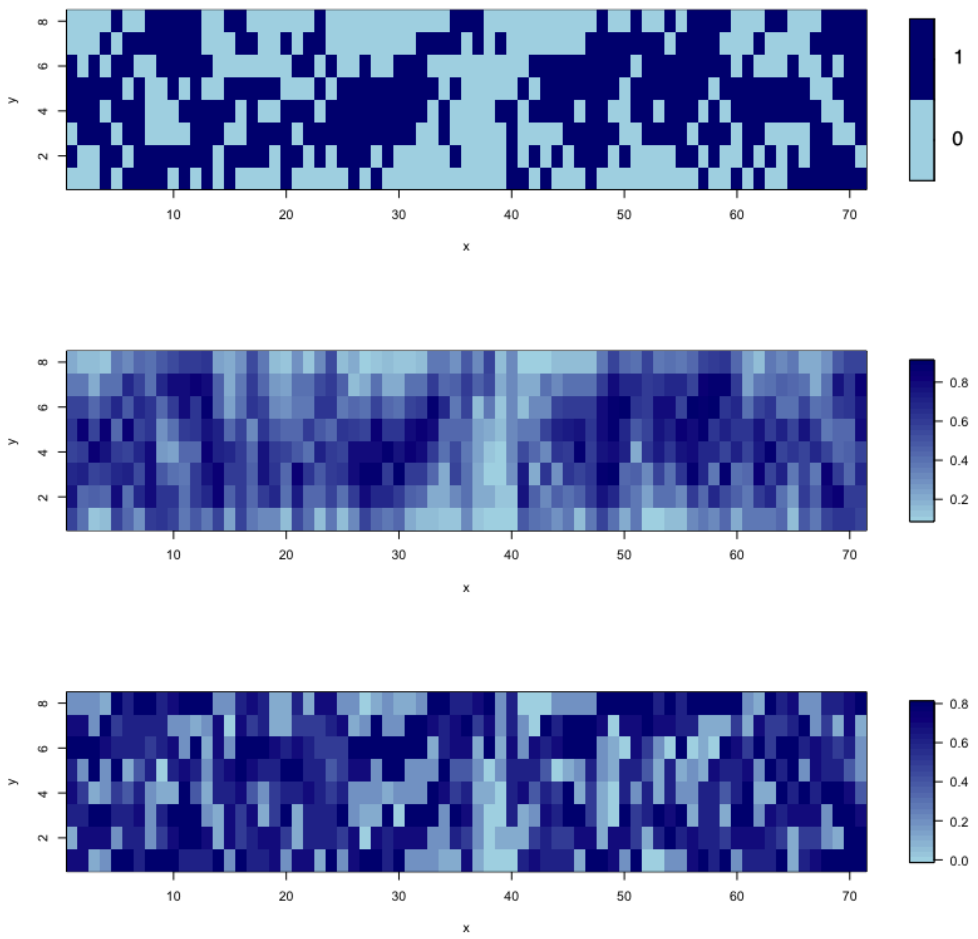



Fig. 5.3 2-d simulation of 0's and 1's from a word length 3 de Bruijn graph that is sticky to both 0's and 1's. The output is shown in the top plot where light blue areas represent a 0 and dark blue areas represent a 1. The middle plot shows the probability of a one, where dark blue indicates high probability. The bottom plot shows the range in probabilities from each direction that are averaged for each letter.

We can clearly see that the number of transition probabilities needed for high dimensions will explode very quickly. Already, we require 158 transition probabilities for the 2-d case when we only have 8 rows in the input grid. Not only do we need the extra two de Bruijn graphs for each dimension, we also require additional transition probabilities depending on how many rows we have in our grid. For the 2-d case, for every increase in row, we require an extra three transition matrices that each have $2^m$ parameters (for word length $m$).

This high quantity of transition probabilities is one of the main disadvantages of this method, and is why I look for other methods to overcome this problem. I am also aware that it is very hard to write down the likelihood for this method due

to the averaging, and so it may not be possible to carry out inference. In the next section, I outline a more successful method that is more inline with the structures of the 1-d de Bruijn process from Chapters 3 and 4.

### 5.2.2 Method 2

Although Method 1 is entirely valid, and I have shown that it is possible to simulate 2-d grids of 0's and 1's with the required structure, it is very hard to take the method any further. Therefore, I propose a new method that has more similarities to the 1-d de Bruijn process, and is a more natural extension.

Retaining the two dimensional grid established in Method 1, we begin to consider potential forms for a 2-d de Bruijn word. With the directional aspect of de Bruijn graphs in mind, I choose the structure given in Figure 5.4. Here, I present the form of the 2-d de Bruijn word for the one, two and three word cases. The points for each word are given by the outlined areas. I decide which points belong to each word by counting how many steps away each point is moving either to the right or upwards. Hence, for the size $m = 1$ word case (red), the word is made up of all the points that can get to the green point in one step by moving either upwards or to the right. This is expanded for the size $m = 2$ word case (purple), where the word consists of all the points that can get to the green point in one or two steps, again only moving upwards or to the right. The pattern also follows for the size $m = 3$ (yellow) and higher word cases.

The structure enforced is now equivalent to the 1-d de Bruijn case due to the 2-d version of the de Bruijn words included. We still require an equivalent to the transition probabilities, but the main difference for the 2-d case is that the concept of transitioning from word to word is still no longer valid. There is still a Markov structure, since each letter is dependent on the word of letters that comes before it, but instead of going from word to word, we can visualise it more as what is the word that the current point is dependent on; i.e., in figure 5.4, what is the word that the state of the green point is dependent on. It is thus no longer possible to visualise the associated graph for this de Bruijn process. As a future problem, it would be

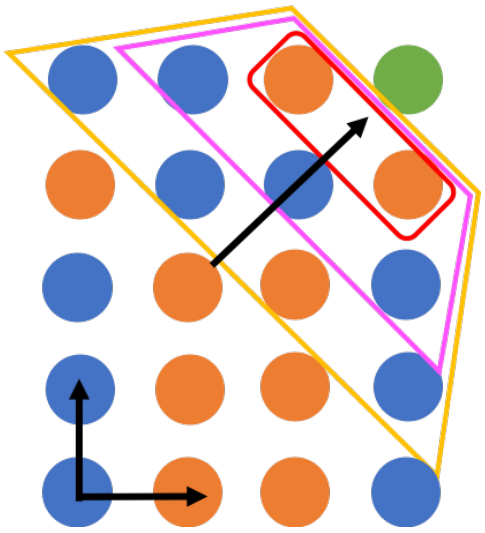


Fig. 5.4 2-d de Bruijn Graph example where the green point is the current point to be simulated. The blue points represent 0's and the orange points represent 1's. The forms of the words for word sizes $m = 1$, $m = 2$ and $m = 3$ that the green point is dependent on are outlined in red, pink and yellow respectively.

interesting to see if we could draw a graph of the possible words each word could transition to. Hence, we would be considering the word to word transitions even though the underlying process transitions from a word to a single letter.

It is clear that the number of transition probabilities associated with each word will get very large with increasing word size. However, it is still possible with enough data and the number of transitions is still greatly reduced as compared with Method 1. There will be $2^{\mu}$ different transition probabilities for each word consisting of $\mu$ letters. The first possible word ($m = 1$) contains $\mu = 2$ letters with $2^2$ transition probabilities, the second word ($m = 2$) contains $\mu = 5$ letters with $2^5$ transition probabilities and the third word ($m = 3$) contains $\mu = 9$ letters with $2^9$ transition probabilities. For an $m$ size word in 2 dimensions, there are $\frac{1}{2}(m^2 + 3m)$ letters in that word with $2^{\frac{1}{2}(m^2+3m)}$ transition probabilities.

For efficiently simulating a grid of 0's and 1's using this 2-d de Bruijn structure, I approach the simulation as if we were in one dimension, so that I can use the tools already developed. To do this, for each word shape, I deconstruct the word to form a vector of 0's and 1's. I can then treat this vector as a 1-d de Bruijn word and carry out the simulation as described in Chapters 3 and 4. A simulation example is presented in Figure 5.5, where we are interested in simulating the letter represented by $*$ given the current simulated 0's and 1's in the grid.

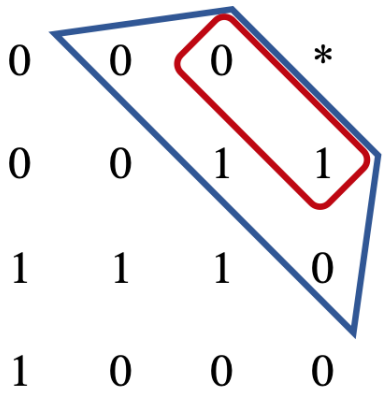



Fig. 5.5 2-d de Bruijn process simulation example where the point represented by $*$ is the current point to be simulated. The forms of the words for word sizes $m = 1$ and $m = 2$ that $*$ is dependent on are outlined in red and blue respectively.

For the $m = 1$ case, $*$ depends on the two closest letters on the diagonal outlined in red. We can treat this 2-d word as the 1-d word equivalent, 10, where we will have an associated probability given this word for what value $*$ will take. The one-word 2-d de Bruijn word is equivalent to the two-word 1-d de Bruijn word. If we now consider the $m = 2$ case, $*$ then depends on the letters in the closest two diagonals. These letters are outlined in blue in Figure 5.5. Converting this 2-d word to a vector, we have the word 01100 with a corresponding probability attached to this regarding the value of $*$. Therefore, the two-word 2-d de Bruijn word is equivalent to the five-word 1-d de Bruijn word. If we were to increase the word length further, we find that an $m$-word 2-d de Bruijn word is equivalent to a $\frac{1}{2}(m^2 + 3m)$-word 1-d de Bruijn word.

Note that the ordering of the letters from each of the 2-d words to its associated vector is arbitrary, but must be maintained for the entire simulation. The main reason for considering the 1-d equivalents for each of these 2-d words is that it makes programming the 2-d simulation simpler, and additional properties are easier to work with and develop. As long as there is a clear one-to-one mapping from each word to its associated transition probability then simulation can be carried out regardless. I use the notation $p_i^1$ to mean the probability of getting a 1 given the dependent word is $i$. The word $i$ will be written in terms of its 1-d equivalent for notational simplicity.

Note the slightly different notation here for the superscript as we no longer transition to whole words, but rather the individual 0 or 1.

One of the main disadvantages for the proposed 2-d de Bruijn process method is the large number of parameters that need to be defined, even for small values of $m$. For $m = 1$ there are 4 transition probabilities, increasing to 32 for $m = 2$ and 512 for $m = 3$. For general word length, $m$, there are $2^d$ parameters, where $d = \frac{1}{2}(m^2 + 3m)$. To limit the number of transition probabilities, we can apply two constraints. The first constraint takes advantage of the known marginal probabilities for the overall proportion of 0's and 1's, allowing us to calculate the transition probabilities from one another. For example, say we knew that 0's and 1's were equally likely, and that the vector of the word was 00101, with transition probability $p$ of getting a 1. This then means that the transition probability of getting a 1 for the vector of the word 11010 is $1 - p$. Hence, we are able to reduce the number of transition probabilities to be estimated by half, becoming $2^{d-1}$.

The other constraint that we are able to make for certain cases takes account of symmetries in the words. If the probabilities in the $x$-direction are known to be the same as the probabilities in the $y$-direction, then we are able to further reduce the total number of transition probabilities. In other words, if we take the words in their 2-d form and one is the reverse of the other in the $x = y$ diagonal, then we can say that the transition probabilities for these words are the same. For a general $m$ length 2-d word, the total number of parameters reduces from $2^d$ to $2^{d-1} + 2^{\frac{1}{4}(2d+m-4-\mathbb{1}_{m\,\text{odd}})}$. In the case that both of these constraints are applicable, we can reduce the total number of parameters to $2^{d-2} + 2^{\frac{1}{4}(2d+m-4-\mathbb{1}_{m\,\text{odd}})}$.

We may also be able to make other constraints to restrict the number of parameters further, but this would require further prior knowledge of the data and the process; i.e., we may have strong priors for certain transition probabilities to help in estimation. For example, if two words have equal numbers of 0's and 1's, then we could use a prior that says they are likely to be very similar. For future work, I could investigate the use of hyperparameters for controlling large numbers of transition parameters.

**Examples**

Figures 5.6 and 5.7 present several examples of simulated 2-d de Bruijn processes, where light blue areas represent the letter 0 and dark blue areas represent the letter 1. Figure 5.6 shows four examples on a grid of size $80 \times 80$. The top two plots are made using $m = 1$ word 2-d de Bruijn processes, and the bottom two plots are produced using $m = 2$ word 2-d de Bruijn processes. The two plots on the left are shown to be highly sticky to both 0's and 1's, and the right two plots are generated using the equivalent to random Bernoulli trials. The effect of adding the de Bruijn structures is clear in both of the left plots, where we can see that the 0's and 1's have started to clump together. We can also see that changing the word size in a 2-d de Bruijn process makes slightly more difference than in a 1-d de Bruijn process. The patches in the $m = 2$ word plot are slightly larger with less distributed random letters than the $m = 1$ word plot. This is likely to be because there are significantly more intricate transition probabilities involved in the $m = 2$ de Bruijn process, allowing a more complex structure to be introduced. There is also a slight diagonal trend in both of these plots. This could be caused either by the word structure or simply the chosen transition probabilities. More research is required to identify this.

For the $m = 1$ word de Bruijn processes shown in Figure 5.6, each word consists of the two previous letters as outlined in red in Figure 5.4. The words are equivalent to $m = 1$ word 1-d de Bruijn words, and hence the four transition probabilities required are: $\{p^1_{00}, p^1_{01}, p^1_{10}, p^1_{11}\}$. For the Bernoulli example, these are all set to 0.5, but for the sticky simulation, I have set these to be: $\{0.1, 0.5, 0.5, 0.9\}$. For the $m = 2$ word plot, we have far more transition probabilities, as the words are equivalent to $m = 5$ word 1-d de Bruijn words. In this case, the words consist of the five previous letters as outlined in purple in Figure 5.4, requiring a total of 32 transition probabilities. Again, I have set all of these probabilities to be 0.5 for the Bernoulli plot, but have set the probabilities to give a sticky result for the other $m = 2$ word plot.

Figure 5.7 gives four examples of 2-word 2-d de Bruijn processes, each of which has a different set of transition probabilities. The top left plot is the most sticky, followed by the top right. Both of these have large concentrated areas of either 0's or

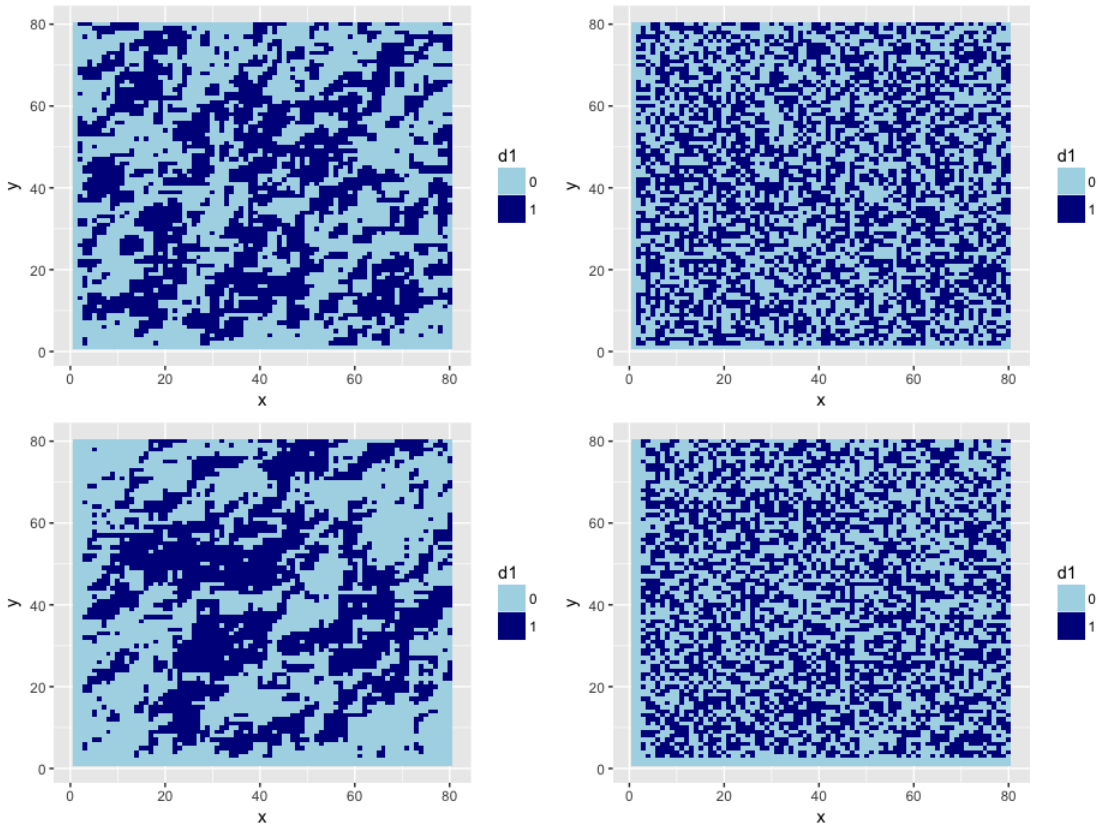



Fig. 5.6 2-d simulations of 0's and 1's from one word de Bruijn processes (top) and two word de Bruijn processes (bottom). The right plots have no correlation structure included (Bernoulli trials), whilst the left plots are shown to have high levels of stickiness for both 0's and 1's.

1's. The bottom left plot is generated using random Bernoulli trials, and the bottom right plot gives a simulation from a de Bruijn process designed to be anti-sticky for both 0's and 1's. This final plot is distinct, in that an almost regular pattern has emerged where the 0's and 1's are almost constantly alternating.

### Inference

Due to the connections with the 1-d de Bruijn process, we can therefore write down the likelihood for the 2-d de Bruijn process, and consider possible inference. The likelihood for the transition probabilities is the same as that for the 1-d version, meaning that the $m = 1$ likelihood has the following form:

$$\begin{aligned}\mathcal{L} &= (p^0_{00})^{n^0_{00}}\,(p^1_{00})^{n^1_{00}}\,(p^0_{01})^{n^0_{01}}\,(p^1_{01})^{n^1_{01}}\,(p^0_{10})^{n^0_{10}}\,(p^1_{10})^{n^1_{10}}\,(p^0_{11})^{n^0_{11}}\,(p^1_{11})^{n^1_{11}}\\ &= (1-p^1_{00})^{n^0_{00}}\,(p^1_{00})^{n^1_{00}}\,(1-p^1_{01})^{n^0_{01}}\,(p^1_{01})^{n^1_{01}}\,(1-p^1_{10})^{n^0_{10}}\,(p^1_{10})^{n^1_{10}}\,(1-p^1_{11})^{n^0_{11}}\,(p^1_{11})^{n^1_{11}},\end{aligned}$$

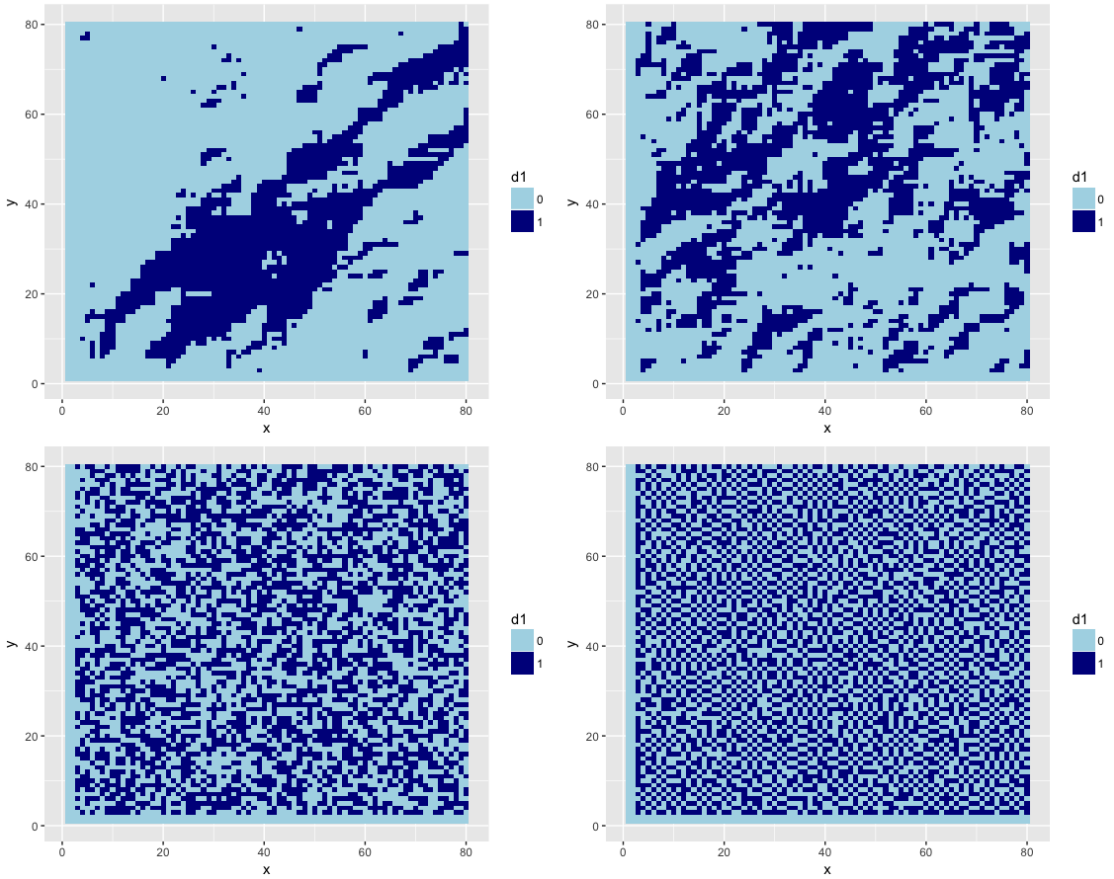



Fig. 5.7 2-d simulations of 0's and 1's from a 2-word de Bruijn processes. The top two plots are shown to have high stickiness towards 0's and 1's, whilst the bottom left plot is generated from random Bernoulli trials. The bottom right plot shows an anti-sticky de Bruijn process.

where $p_i^1$ is the probability of getting a 1 given that the dependent word is $i$, and $n_i^1$ is the number of times that transition takes place. The transition likelihood for general $m \geq 1$ cases is given by:

$$\mathcal{L} = \prod_{i=0}^{2^d-1} \left(1 - p_i^1\right)^{n_i^0} \left(p_i^1\right)^{n_i^1}$$

where $d = \frac{1}{2}\left(m^2 + 3m\right)$. This is written in terms of the decimal representation of the binary values, as was done for the 1-d version.

Since the general likelihood is equivalent to the 1-d de Bruijn process in Theorem 4.20, we can therefore use the exact same method of inference to estimate the transition probabilities. Further, we can use the method of Bayes' factors (Kass and Raftery, 1995; O'Hagan, 1997) from Theorem 4.22 to best estimate the word length

for a given sequence. I will now present two examples to show how this inference method is still effective for the 2-d de Bruijn processes.

The first example is shown in Figure 5.8, which is a simulation on an $80 \times 80$ grid from a 2-d de Bruijn process with word length $m = 1$ and transition probabilities: $\{p^1_{00}, p^1_{01}, p^1_{10}, p^1_{11}\} = \{0.1, 0.5, 0.5, 0.9\}$. Given that I know the data was generated using a 1-word de Bruijn process, the transition probabilities are estimated using the 1-d inference described in Section 4.3. This can either be done using a maximum likelihood approach or by using a simple Metropolis Hastings MCMC. I treat the estimation as a 2-word 1-d problem, and the results are given in Table 5.1. I have also included estimates from two other sized grids ($30 \times 30$ and $150 \times 150$) to show how differently sized data effects the estimation of the transition probabilities. Both of these examples were generated using the same set of transition probabilities.

We can see that the method has successfully estimated the transition probabilities for all three simulation grid sizes, where as expected, the larger two sizes have a higher level of accuracy. The average error for each of sizes are 10.44%, 2.52% and 1.03% respectively.

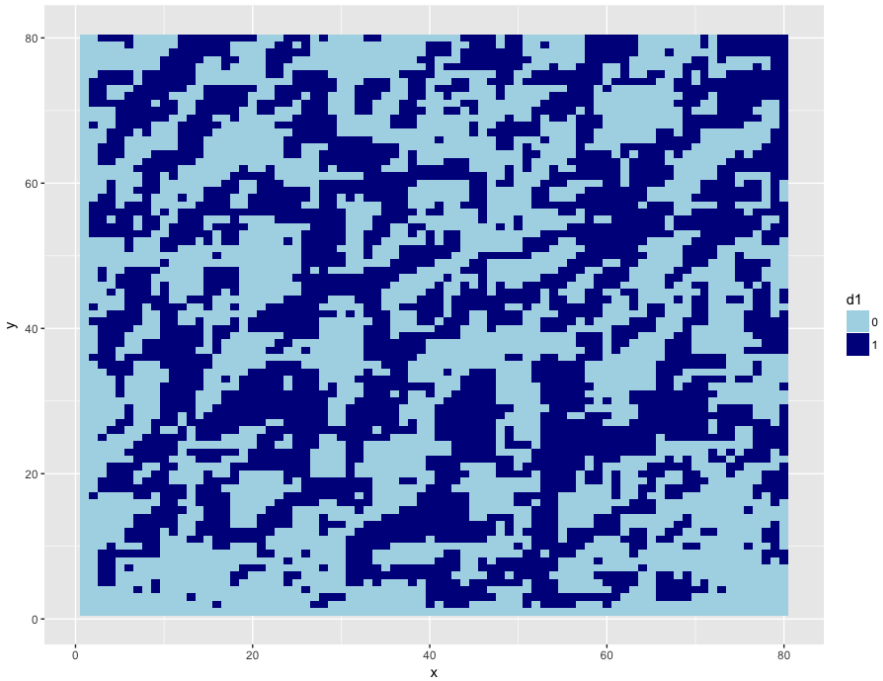



Fig. 5.8 2-d simulation of size $80 \times 80$ consisting of 0's and 1's from a 2-d de Bruijn processes with word length $m = 1$ and transition probabilities: $\{p^1_{00}, p^1_{01}, p^1_{10}, p^1_{11}\} = \{0.1, 0.5, 0.5, 0.9\}$.

| $p$ | $30 \times 30$ | $80 \times 80$ | $150 \times 150$ |
|---|---|---|---|
| 0.1 | 0.123 | 0.108 | 0.098 |
| 0.5 | 0.537 | 0.497 | 0.493 |
| 0.5 | 0.544 | 0.506 | 0.498 |
| 0.9 | 0.923 | 0.897 | 0.903 |

Table 5.1 Table to show estimates of the transition probabilities from the example in Figure 5.8. The true values, $\{p^1_{00}, p^1_{01}, p^1_{10}, p^1_{11}\} = \{0.1, 0.5, 0.5, 0.9\}$, are shown on the y-margin. Estimates are given for data grid sizes $30 \times 30$, $80 \times 80$ and $150 \times 150$.

The second example is given in Figure 5.9, where I estimate both the word length and the transition probabilities. The simulation is an $80 \times 80$ grid of 0's and 1's generated from a 2-word 2-d de Bruijn process, where the transition probabilities are set to be sticky for both 0's and 1's.

I first attempt to estimate the most likely word length used to generate the example, using the method of Bayes' factors with the model evidence stated in Theorem 4.22 for the 1-d methodology. I start by calculating the model evidence for the 2-d grid for four different models corresponding to 2-d de Bruijn processes with words $m = 1$, $m = 2$, $m = 3$ and $m = 4$. These are equivalent to 1-d de Bruijn processes with word lengths $m = 2$, $m = 5$, $m = 9$ and $m = 14$ respectively, and so we generate the model evidences for the given data as if they were 1-d de Bruijn processes of these lengths. To calculate these, we require the prior distributions for the transition probabilities. I do not assume that we have any prior knowledge of the transitions and hence let $\alpha = \beta = 1$ in Equation 4.22 for the equivalence of a uniform prior. I then calculate a Bayes' factor for each pair of word lengths such that:

$$B_{i,j} = \frac{P(S|m_i)}{P(S|m_j)}, \qquad \text{for} \quad i, j = 1, ..., 4.$$

The values for $B_{i,j}$ for $i, j = 1, ..., 4$ are given in Table 5.2. When $B_{i,j} > B_{i',j}$ for all $i' = 1, ..., 4$ and $j = 1, ..., 4$, then we can conclude that the grid of 0's and 1's was most likely created with a length $m = i$ de Bruijn process. Looking at the table, we can see that this is the case for the second column and therefore the method suggests that the simulation data comes from an $m = 2$ word 2-d de Bruijn graph.

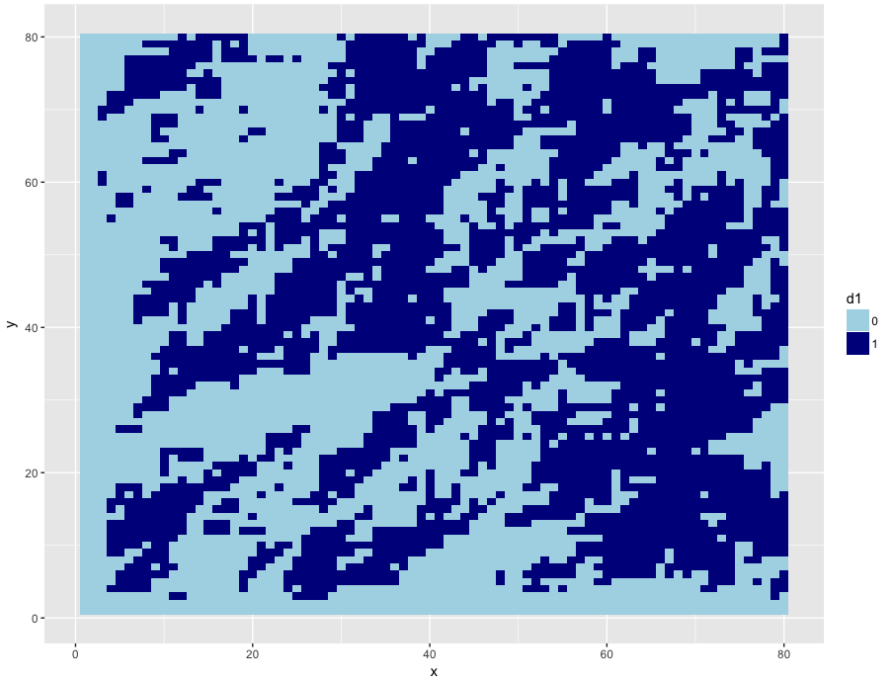



Fig. 5.9 2-d simulation of size $80 \times 80$ consisting of 0's and 1's from a 2-d de Bruijn processes with word length $m = 2$. The transition probabilities for this example are given in Table 5.3.

| $m$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0.00 | 229.63 | -63.91 | -1066.21 |
| 2 | -229.63 | 0.00 | -293.54 | -1295.84 |
| 3 | 63.91 | 293.54 | 0.00 | -1002.31 |
| 4 | 1066.21 | 1295.84 | 1002.31 | 0.00 |

Table 5.2 Table giving the log Bayes' factors for 4 models with 2-d word lengths, $m = 1, 2, 3, 4$ for the given data in Figure 5.9. These are equivalent to the 1-d word length, $m = 2, 5, 9, 14$.

Having estimated the word length to be $m = 2$, I now estimate the corresponding transition probabilities. A 2-word 2-d de Bruijn word is equivalent to a 5-word 1-d de Bruijn word, giving a total of 32 transition probabilities to be estimated. Using the simplifying conditions described above we can reduce the number of parameters. First, by using the marginal probabilities for the overall numbers of 0's and 1's, we can reduce the total number by half to 16. The number of parameters can then be further reduced by taking account of the symmetries in words, making the total number of transition probabilities to be estimated 10.

Given the reduced number of parameters, I take a Bayesian approach with uninformative priors to estimate the transition probabilities. The true transition

probabilities along with estimates for each number of parameters (32, 16 or 10) are given in Table 5.3. $p$ is the true value for the transition probability given by $p_i^1$ and $\hat{p}$ is the corresponding estimate. We can see that although the method does fairly well at estimating all of the transition probabilities, the estimates when there are far fewer parameters are superior. I therefore recommend to simplify the number of parameters used where possible.

### 5.2.3 3 Dimensions and Higher

Overall, it is clear that Method 2 is preferable as it has a structure that closely resembles the successful structure of the one dimensional de Bruijn process. It also requires far fewer parameters to build sticky 2-d grids of 0's and 1's, and we are able to perform inference on both the word and the transition probabilities. An obvious extension is to now expand this method to higher dimensions. Given that we are able to reduce the 2-d de Bruijn process down to something that looks and behaves like a 1-d de Bruijn process, we can therefore consider doing the same for higher dimensions. We would then be able to continue using the same methods for simulation and inference from the 1-d methodology, with the only difference being the shape and form of the word. As long as we are able to reduce these words down to a 1-d vector, then there is no reason why we cannot form $n$ dimensional de Bruijn processes.

Let us first consider 3-d de Bruijn processes, where, to form the words, we can take a similar approach to that of the 2-d case. The word structure on a 3-d grid is shown in Figure 5.10. For an $m$-word, we must include all the points that are $m$ moves away from the point of interest, only moving in the up, forward and right directions.

From Figure 5.10, we can see that the 3-d words form shapes similar to corners from a cube. To form the next larger word, the corner increases in size by the addition of the next triangle plane. The first word (green) contains three points, and so is equivalent to the 3-word 1-d de Bruijn word. The second word (blue) contains 9 points, and is thus equivalent to the 9-word 1-d de Bruijn word. This

| $p_i^1$ | $p$ | $\hat{p}$ |
|---|---|---|
| $p_0^1$ | 0.05 | 0.054 |
| $p_1^1$ | 0.10 | 0.091 |
| $p_2^1$ | 0.40 | 0.429 |
| $p_3^1$ | 0.50 | 0.508 |
| $p_4^1$ | 0.15 | 0.197 |
| $p_5^1$ | 0.20 | 0.124 |
| $p_6^1$ | 0.50 | 0.585 |
| $p_7^1$ | 0.50 | 0.473 |
| $p_8^1$ | 0.40 | 0.362 |
| $p_9^1$ | 0.55 | 0.635 |
| $p_{10}^1$ | 0.75 | 0.716 |
| $p_{11}^1$ | 0.80 | 0.898 |
| $p_{12}^1$ | 0.55 | 0.536 |
| $p_{13}^1$ | 0.55 | 0.651 |
| $p_{14}^1$ | 0.80 | 0.805 |
| $p_{15}^1$ | 0.85 | 0.874 |
| $p_{16}^1$ | 0.15 | 0.152 |
| $p_{17}^1$ | 0.20 | 0.203 |
| $p_{18}^1$ | 0.45 | 0.437 |
| $p_{19}^1$ | 0.45 | 0.496 |
| $p_{20}^1$ | 0.20 | 0.174 |
| $p_{21}^1$ | 0.25 | 0.198 |
| $p_{22}^1$ | 0.45 | 0.466 |
| $p_{23}^1$ | 0.60 | 0.599 |
| $p_{24}^1$ | 0.50 | 0.456 |
| $p_{25}^1$ | 0.50 | 0.452 |
| $p_{26}^1$ | 0.80 | 0.876 |
| $p_{27}^1$ | 0.85 | 0.831 |
| $p_{28}^1$ | 0.50 | 0.489 |
| $p_{29}^1$ | 0.60 | 0.610 |
| $p_{30}^1$ | 0.90 | 0.900 |
| $p_{31}^1$ | 0.95 | 0.947 |

| $p_i^1$ | $p$ | $\hat{p}$ |
|---|---|---|
| $p_0^1$ | 0.05 | 0.155 |
| $p_1^1$ | 0.10 | 0.110 |
| $p_2^1$ | 0.40 | 0.395 |
| $p_3^1$ | 0.50 | 0.498 |
| $p_4^1$ | 0.15 | 0.172 |
| $p_5^1$ | 0.20 | 0.221 |
| $p_6^1$ | 0.50 | 0.522 |
| $p_7^1$ | 0.50 | 0.463 |
| $p_8^1$ | 0.40 | 0.421 |
| $p_9^1$ | 0.55 | 0.494 |
| $p_{10}^1$ | 0.75 | 0.809 |
| $p_{11}^1$ | 0.80 | 0.721 |
| $p_{12}^1$ | 0.55 | 0.579 |
| $p_{13}^1$ | 0.55 | 0.535 |
| $p_{14}^1$ | 0.80 | 0.800 |
| $p_{15}^1$ | 0.85 | 0.848 |

| $p_i^1$ | $p$ | $\hat{p}$ |
|---|---|---|
| $p_0^1$ | 0.05 | 0.050 |
| $p_1^1$ | 0.10 | 0.102 |
| $p_2^1$ | 0.40 | 0.374 |
| $p_3^1$ | 0.50 | 0.499 |
| $p_4^1$ | 0.15 | 0.105 |
| $p_5^1$ | 0.20 | 0.198 |
| $p_6^1$ | 0.50 | 0.493 |
| $p_9^1$ | 0.55 | 0.555 |
| $p_{10}^1$ | 0.75 | 0.758 |
| $p_{14}^1$ | 0.80 | 0.818 |

Table 5.3 Tables to show estimates of the transition probabilities from the example in Figure 5.9. In each table, $p_i^1$ shows the transition probability (using the decimal representation of the binary word), $p$ gives the true transition probability and $\hat{p}$ gives the estimate for the transition probability. The far left table gives the estimates for all 32 parameters, whilst the other two tables give estimates for parameters where constraints have been made (16 and 10 parameters respectively).

pattern continues, where the 3-word 3-d de Bruijn word is equivalent to the 19-word 1-d de Bruijn word and the $m$-word 3-d de Bruijn word is equivalent to the $\frac{1}{6}(m^3 + 6m^2 + 11m)$-word 1-d de Bruijn word.

It is also important to note that the set of additional points included with each increase in word size is equivalent to the same word of the dimension below (making

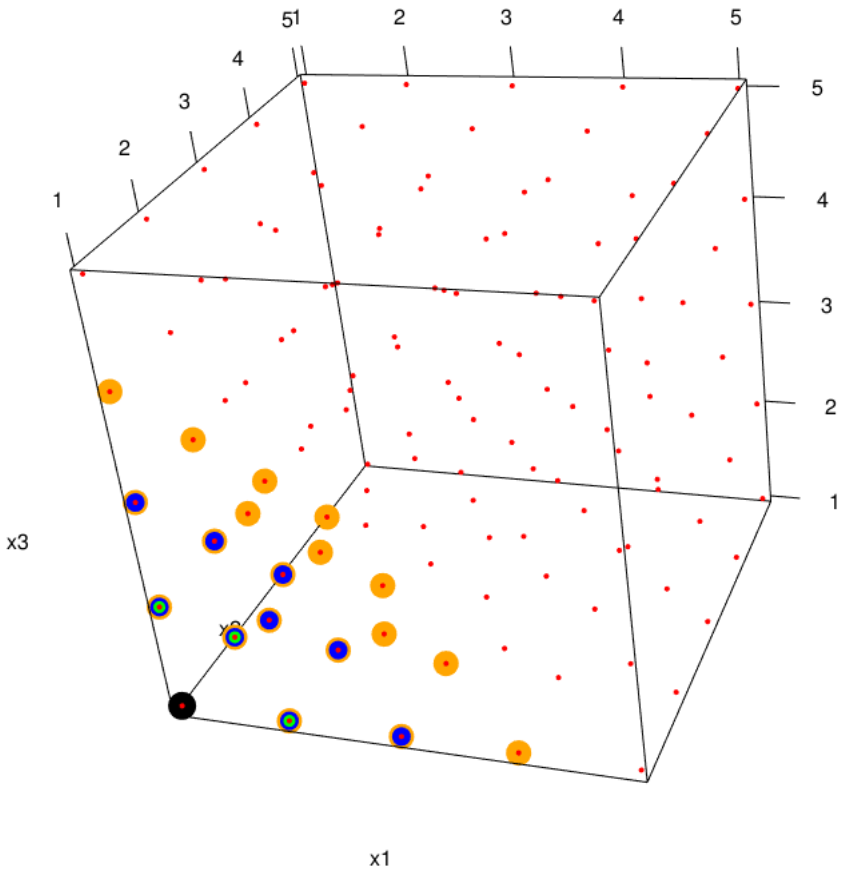



Fig. 5.10 Plot to show the form of de Bruijn words in 3 dimensions. The words that the black point is dependent on are shown in green ($m = 1$), blue ($m = 2$) and orange ($m = 3$).

sure we include the extra point). In other words, the triangle that is added to the 2-word for the 3-word 3-d de Bruijn (only orange points in Figure 5.10) is the same as the 3-word 2-d de Bruijn (including 1 extra point). This relationship appears to be the same regardless of the dimension or word size, meaning that we can form the following relationship for the number of points in the word for an $m$-word $n$-d de Bruijn graph:

$$a_{n,m+1} = a_{n,m} + a_{n-1,m+1} + 1,$$

where, $a_{n,m}$ is the number of points in the $m$ size de Bruijn word in $n$ dimensions.

As the dimension increases, it is clear that the number of parameters will again explode. Hence, it becomes increasingly important to use constraints similar to those found for Method 2. It is possible to reduce parameters by considering the marginal distributions for each letter, and by observing any symmetries in the words

themselves. We can also make other assumptions, such as defining certain transition probabilities to be equal, or by providing strong priors in estimation. Examples in higher dimensions are left for future work.

### 5.2.4 Run Length and Future Work

One of the key results that I focused on in Chapter 4 was the run length distributions of letters in a sequence obtained from a de Bruijn process. In this section, we begin to consider what the two dimensional equivalent of a run length is. This is a hard problem, since we do not naturally have a start and end point to a run of letters that we had in the 1-d version. Instead of a run of 1's, we must consider what a patch size or area of 1's would be defined as on our grid.

To begin with, we could define the patch size by counting how many 1's (or 0's) are touching at least one other 1 (or 0). However, this means that we can end up with long chains of 1's that would technically be equivalent to a more circular patch that contained the same number of 1's. Therefore, we arrive at the situation where it depends on what one defines as a patch. If we require patches that are spherical, then we could require that each 1 must be touching at least two other 1's on the grid to be classed as a patch. Equally, we may consider how many 1's are touching in small groups, such as a cross made up of five individual 1's. Hence, it may be necessary to consider what is suitable for a specific application before defining a constraint on the patch shape.

In relevant literature, the concept of patch size has been discussed in mathematical ecology (Pielou, 1969, 1984). For example, we may want to model patch size of trees in a forest, or estimate the maximum size for a collection of bacteria in an experiment. Pielou (1969) discusses the measurement of aggregation of a population's spatial pattern. The result of his work is essentially what I require, as I wish to know how clustered each letter is in a defined grid of 0's and 1's: the author describes several methods to do this. The first of his methods assumes that the data is Poisson distributed, and so he considers the ratio between the data variance and mean to test for randomness. Highly clustered discrete distributions tend to have a variance

which exceeds the mean. Hence, if the points are dispersed at random, the ratio of the variance and mean should be close to one. With this in mind, I may want to consider patch size in random Poisson processes before focusing on the 2-d de Bruijn process.

Secondly, he references Lloyd's indices of mean crowding and patchiness as a measure of aggregation. For each individual point, this requires defining the number of other points that are in the same pre-defined neighbourhood. Hence, we can calculate the mean number per individual point of other individual points in a neighbourhood.

It is clear that patch size, and measurements of clustering in general, is a large topic itself. Hence I make the choice to leave this as future work. I believe the main focus of such future work should be developing a formal definition of a patch size. Although the concepts described in the ecological literature are interesting, and I can use some of the main ideas, they are mostly focused on testing whether clustering exists instead of actually modelling it.

As well as patch size, we could consider diagonal lines of 0's and 1's across the grid as an alternative to run length. In the setup of the word structure described in Method 2, I ended up building the grid across diagonal lines from the top left hand corner down to the bottom right hand corner. However, this means that I do not take account of the correlation between neighbouring points along this diagonal. There must be some degree of dependence structure along these diagonals, as the points along the diagonal are dependent on several of the same previous points in their respective words. To see this correlation, we could consider the distribution or expected run length of the letters along the diagonals. This would provide more information on the connections between these neighbouring points on the diagonal line. I may then be able to consider a de Bruijn structure along these diagonal lines, so that we have each line being dependent on the lines before. I can also analyse the structure and correlation across each of these diagonal lines. Such an analysis would be far more difficult than when we considered the run lengths in the one dimensional case, as we do not have the de Bruijn structure along the diagonals; i.e., we do not

know which letter comes before in the diagonal until we actually simulate it. Hence, I leave this for future work.

As mentioned throughout this section, the major hurdle of de Bruijn processes in higher dimensions is the directionality. Despite my attempts to produce a two dimensional de Bruijn process in this chapter, the directionality in a 2-d grid still does not make logical sense. Inspired by the work from Wolfram (2002) and Besag (1986), Chapter 6 explains my initial thoughts on how to develop a de Bruijn process with the directionality removed. This is a very difficult problem to tackle, hence I have not completed all work proposed, where much of it is left for future research.

## 5.3 Discussion

In this section I have presented an introduction to a two dimensional de Bruijn process. The main problem to overcome with this was the natural directionality of de Bruijn graphs. I can no longer start at one end and let the de Bruijn graph run to produce a sequence of 0's and 1's. Instead, I consider trying to produce a grid of 0's and 1's that lies in two dimensions with the hope that I can easily extend the method to higher dimensions.

Since I used Markov chains in the one dimensional de Bruijn process so that the words had a Markov property, I initially began thinking about 2-d alternatives including Markov random fields, Markov mesh grids and CAR models. I also discussed the possibility of using a version of cellular automata to iteratively produce grids or lattices of 0's and 1's. Since these methods had a lack of directionality, I decided to focus on these in a later chapter, and first attempted a directional 2-d de Bruijn process. Although I believe that a non-directional de Bruijn process will be a vital way forward with the correlated Bernoulli process, I was keen to see if there was a natural 2-d extension to the already developed 1-d directional de Bruijn process.

Progressing on from here, I began developing my own methods for a directional 2-d de Bruijn process. My first method (Method 1) has connections to multivariate and higher-order Markov chains which average over several sequences that have been generated from a similar source. I took this concept and combined it with the de

Bruijn word structure to create a de Bruijn process. Depending on an $m$ length de Bruijn graph, I generated a simulation by averaging over the closest $m$ rows above and below for each point in a grid. Hence, for simulating each point in the grid, I averaged over three separate but linked de Bruijn graphs across the same row and the upper and lower diagonals. I therefore required a transition matrix (similar to those in Sections 3 and 4) for each row, upwards diagonal and lower diagonal in the grid I was simulating. Since this generated a large number of parameters in the model, I chose not to progress any further in this way. I also found that developing any inference for this method would be challenging due to the averaging that takes place at each step.

I then moved on to improve on this with my second method (Method 2). My aim when developing this method was to consider what the two dimensional equivalent to a de Bruijn word is. By doing this, I hoped to produce a method that was more similar to the one dimensional version so that parts of my 1-d methodology could be carried over to the 2-d version. I therefore defined the form of the 2-d word structure in Figure 5.4. However, I found that since we can no longer transition from word to word (instead we have word to letter), we can no longer visualise the associated graph to the process.

The major advantage of this method is that we can convert each of the 2-d words into their 1-d word equivalent. For example, a 1 word 2-d de Bruijn word is equivalent to a 2 word 1-d de Bruijn word and a 2 word 2-d de Bruijn word is equivalent to a 5 word 1-d de Bruijn word. Providing we keep the ordering consistent, we can convert each 2-d word into a vector and treat it as if it were the 1-d equivalent. This means that we can use the same method of simulation from the 1-d de Bruijn process, but more importantly it enables us to perform inference on 2-d de Bruijn processes. Then with little extra work, we can convert this method into higher dimensions to produce an $n$-d de Bruijn process. For future work, I plan to find the equivalence of the run length distribution from Chapter 4 for 2-d de Bruijn processes.

It has become clear that a non-directional version of a de Bruijn graph would be ideal in simulating two dimensional (and higher) grids of 0's and 1's for my correlated

Bernoulli process. Although it is a challenging problem, I present my initial thoughts on the topic in the next chapter.

# Chapter 6

# Towards Non-directional De Bruijn Graph Structures

## 6.1 Introduction

One of the major disadvantages of the current method for de Bruijn processes (Chapters 3, 4 and 5) is the directionality that is required. This is especially the case when trying to form a 2-d version of a de Bruijn process: we are trying to force a specific directionality on a spatial grid where directionality does not make logical sense. In this chapter, the aim is to find a non-directional equivalent to the de Bruijn process.

The inspiration for this came from looking at Besag (1986). Here, the author develops a new method for reconstructing two-dimensional images, where the aim is to estimate the true colouring for each pixel located in the image. The image (region) is split up into a fine rectangular grid, where each of the pixels in this grid has a true colouring from a set of known colours. The main application for the work is to reconstruct blurred or damaged images, hence the method is required to have a starting image for reconstruction.

Their model combines two sets of data. One is a multivariate record containing some information of the colouring at each pixel, which follows a known statistical model (normally Gaussian). The other data source states that pixels close together tend to have the same colour, and is presented as a non-generate Markov random

field. We can relate these ideas to the de Bruijn process by having a set of two possible colours (i.e. black = 1 and white = 0) which are dependent on a specified local neighbourhood of points. If we were to adapt this method to be more suited to de Bruijn processes, we could consider including an extra dependency on the size of the neighbourhood, changing how many points are included. We would also need to define what the initial image would be, and consider how to define the information included in the statistical model. The author uses a Normal distribution, but we could investigate whether a binary distribution could also be used.

For inference, Besag (1986) makes use of Bayes' theorem to find the probability of any colour at each pixel. The likelihood becomes an observation from a Gaussian distribution, and the prior is expressed as a simple Markov random field defining that neighbouring points will be similar in colour.

As discussed in Chapter 5, there are also other alternative methods in the Markov random field literature. These include CAR models (Banerjee et al., 2004; Besag, 1974) and Markov meshes (Abend et al., 1965). Both of these act in a similar way to the work by Besag (1986) as each point in a specified grid is dependent on the local neighbourhood of points. Cellular automata (Agapie et al., 2014, 2004; Wolfram, 2002) were also mentioned in Chapter 5, where (again) the value of a particular cell in a grid depends on a set of rules based on the neighbouring cells.

Combining the ideas of the above literature and the de Bruijn process already developed, I will attempt to produce a method for a non-directional de Bruijn process. This is a difficult problem, and the majority of the research for this is left for future work. I begin by attempting to define a non-directional de Bruin process for the one-dimensional case in Section 6.2. Due to the success of converting the 2-d directional de Bruijn words into their 1-d word equivalents in Chapter 5 (Method 2), I also attempt this for the non-directional 1-d de Bruijn words. From this, we will then be able to calculate a run length distribution (Section 6.2.2) with expectation, variance and generating functions. In Section 6.2.9, I then show how it is possible to apply the same method of inference from Chapter 4 using Bayes' factors. Section 6.3

outlines my initial ideas on a 2-d non-directional de Bruijn process (the majority of this is left for future work).

## 6.2 1-d Methodology

The main aim for this section is to develop a method for a de Bruijn process that has many of the same characteristics as the existing 1-d de Bruijn process, but is no longer restricted to a specific direction. As shown in Chapter 5, it was possible to retain some of the structure from the 1-d de Bruijn process for the 2-d de Bruijn process (Method 2), meaning that I could use some of the existing properties developed, such as inference. Therefore, I choose to keep the idea of 'words' which are vital to de Bruijn graphs, but change the form.

Previously, in the 1-d directional de Bruijn process, the word consisted of the $m$ letters that came before the point of interest. Here, to ensure that the dependency is equivalent either side of each point, I define the word of the de Bruijn process to now consist of the $m$ letters that fall on both sides of the point of interest. This ensures that there is no direction involved in the dependent word. Since there are equal numbers of influential points on each side, the total number of possible words for each word length $m$ is $2^{2m}$. Examples on non-directional words are given in Figure, 6.1, where the $m = 1$, $m = 2$ and $m = 3$ word cases that the green point is dependent on are labelled. Here, the blue points represent 0's and the orange points represent 1's. The one-word case is shown in red, the two-word is shown in purple, and the three-word is shown in yellow.

Fig. 6.1 1-d non-directional de Bruijn example to show the form of specific words. Blue points correspond to 0's and orange points correspond to 1's. The green point is the point of interest for looking at which words it is dependent on. The forms of the word for m=1, m=2 and m=3 are outlined in red, purple and yellow respectively.

The classification of a single point being either a 0 or a 1 is dependent on the number and positions of the $m$ 0's and 1's located either side of the point, forming

the word. As in directional de Bruijn processes, each of the possible words can have a probability attached to them. Since this will tell us the probability of the centre point (green point in Figure 6.1) being a 1, we find that it is no longer possible to transition to full words, but rather just to a single 0 or 1. To avoid confusion, these probabilities will now be known as conditional word probabilities. They are written as $p^1_{i:j}$, representing the probability of getting a 1 from the word $i:j$. For example, if we had the sequence $100*101$, then the probability of $*$ being a 1 for the one-word case would be displayed as $p^1_{0:1}$ since $*$ has a 0 to the left and a 1 to the right. For the two-word and three-word cases, the probabilities of $*$ being a 1 would be $p^1_{00:10}$ and $p^1_{100:101}$ respectively. Therefore, in a similar way to Method 2 in Chapter 5, we are also not able to easily visualise the associated de Bruijn graph for this process.

Using these conditional word probabilities, we are still able to influence the 'stickiness' of the generated sequences from the de Bruijn process. If the word consists of entirely 1's (or 0's), and the probability of the centre point also being a 1 (or 0) is close to one, then we would expect this sequence to be very sticky. For example, if $m = 3$, then for the sequences generated from the de Bruijn process to be sticky to both 0's and 1's, we would expect the probabilities $p^0_{000:000}$ and $p^1_{111:111}$ to be close to one.

Even though we cannot consider the word to word graph of the non-directional de Bruijn process, and hence cannot form a transition matrix, we can still use some of the same identities from the directional process. The most important of these is the necessity that the rows of the transition matrix sum to one. For example, if $m = 3$, then $p^0_{000:000} + p^1_{000:000} = 1$. This is also true for any word of length $m$. This relationship still holds true as it is only possible to produce either a 0 or a 1 from each conditioning word, hence the sum of the conditional word probabilities of getting either a 0 or a 1 must be equal to one. Unfortunately, it is no longer possible to easily find Markov properties such as the stationary distribution or the convergence rate as in Chapter 3, and thus is left for future work. One future problem is to decide whether the process is Markov or not, as it is not clear from initial observations. The conditional independence assumption still holds since each symbol is only dependent

on the closest word of symbols either side and no other. However, since we are no longer transitioning from word to word, we no longer have a clear set of states for the associated Markov chain. To prove that the process is Markov, it must satisfy all of the conditions from the Markov property definition in Section 3.2. It may be that we have to change which part of the process has a Markov property. For example, we could treat an entire sequence as one state and observe the Markov property as we transition from one sequence to the next. Alternatively, we may have a collection of Markov chains that describe the process, so that each symbol and associated word becomes a single Markov chain. I do not fully understand the process at this level, so it is something that I am keen to progress with in future research. If we can prove that the process is Markov in some way, then we will be able to use well known properties to help progress the methodology further.

As described above, it would be ideal if the non-directional de Bruijn process had a similar form to the 1-d directional de Bruijn process so that the properties and inference developed for this could be carried over. For the 2-d de Bruijn process in Method 2, we were able to decompose each of the 2-d de Bruijn words into their 1-d de Bruijn word equivalents. Hence, we were still able to use the method of Bayes' factors with either maximum likelihood or Bayes' theorem to estimate both the word length and the associated transition probabilities. All that is necessary for this is to have a one-to-one corresponding transition probability for each possible word.

Each of the non-directional words are written as $i : j$. If we convert $i : j$ to its vector form, $ij$, then it becomes the 1-d directional de Bruijn word equivalent to the non-directional word. Since each non-directional word can be written as its directional word equivalent, I can conclude that we can also view this problem in the same way as the 1-d directional de Bruijn process. Thus, a large part of the 1d directional de Bruijn process methodology can be used for the non-directional de Bruijn process. By having a one-to-one correspondence from the non-directional word $i : j$ to the directional word $ij$, I can conclude that a length $m$ non-directional de Bruijn word is equivalent to a length $2m$ directional de Bruijn word.

I have chosen to display a non-directional word in the form $i : j$ since it is easier to interpret the sequence that we are referring to, rather than writing the split word as one. In the calculations that follow, the words will be written in terms of their decimal representations of the binary values, as used previously.

### 6.2.1 Simulation and Examples

One major difference between the non-directional and the directional de Bruijn processes is the simulation. In the directional version, a sequence of 0's and 1's was generated by letting the Markov chain on the words run for the desired length. At each step, the current letter was dependent on those that came before in the set word length neighbourhood. For the non-directional version, since each letter is dependent on those located either side, this is no longer possible. In this section, I describe two possible alternate methods for simulating a sequence from a non-directional de Bruijn process.

When simulating a sequence for the directional de Bruijn process, we required the first $m$ letters to be known (and fixed) to act as an initial condition. This initial condition is also required for the non-directional version, but we now require $m$ initial known letters at both ends of the sequence. This condition enforces that the process is now truly non-directional. We can arbitrarily define these initial letters, or use reduced word length de Bruijn processes to simulate the $m$ edge letters. At least two fixed boundary letters (one at each end) will be required for each simulated sequence. Alternatively, to avoid any such boundary condition complications, and to show examples for the methods that will follow, we can form the sequence into a loop so that the de Bruijn condition affects both of the end letters (periodic boundary conditions).

#### Simulation 1

The joint probability of obtaining a sequence of 0's and 1's can be defined as the product of conditional word probabilities. This is possible because the product of probabilities defines the exact ordering of the words for the sequence, and hence it

also uniquely defines the sequence of letters. One possible method for simulation I will discuss uses this definition of the joint probability to iteratively produce possible sequences given the de Bruijn process. At each time step, we calculate the joint probability of obtaining every possible sequence given the current sequence, then draw from these to give a single realisation from the process.

First, begin with an initial estimate for each of the letters, $v$, in a length $n$ sequence: $(v_1^{(0)}, v_2^{(0)}, ..., v_n^{(0)})$. This sequence can then be written in terms of its non-directional de Bruijn words: $(w_1^{(0)}, w_2^{(0)}, ..., w_n^{(0)})$. To avoid boundary condition problems, we form the sequence into a loop, so that the end letters are dependent on the letters at the opposite end of the sequence (periodicity). For each iteration, $i = 1, 2, ...$, we calculate:

$$\mathbf{P}^{(i)} = p^{j_1}_{w_1^{(i-1)}} \times p^{j_2}_{w_2^{(i-1)}} \times ... \times p^{j_n}_{w_n^{(i-1)}}$$

for $j_k \in \{0, 1\}$ and where $p^{j_k}_{w_k}$ is the probability of obtaining the letter $j_k$ from the non-directional word $w_k$. $\mathbf{P}^{(i)}$ is then a vector of joint probabilities for each possible sequence of 0's and 1's obtained from the $(i-1)^{\text{th}}$ sequence at iteration $i$. Drawing from $\mathbf{P}^{(i)}$ gives us the $i^{\text{th}}$ sequence in our iterations, which is the most likely given both the current sequence and the conditional word probabilities. This process is repeated a number of times until convergence of a suitable sequence, giving a sample from the de Bruijn process.

Three samples from non-directional de Bruijn processes with word length $m = 1$ are given in Figure 6.2. The marginal probabilities are $\pi(\{0\}) = \pi(\{1\}) = 0.5$ for each sequence and the conditional word probabilities $\{p^1_{0:0}, p^1_{0:1}, p^1_{1:0}, p^1_{1:1}\}$, are: $\{0.5, 0.5, 0.5, 0.5\}$, $\{0.1, 0.1, 0.9, 0.9\}$ and $\{0.1, 0.9, 0.1, 0.9\}$ respectively (from top to bottom). The 0's and 1's in the top plot appear uncorrelated as expected, as this sequence is equivalent to drawing independently from a Bernoulli distribution with probability 0.5. The bottom two sequences are seen to be more sticky with larger groups of both 0's and 1's, which is also to be expected as these de Bruijn processes were designed to be equally sticky.

Although this method of simulation appears to be successful, it is very computationally inefficient. For a length $n$ sequence, we must calculate and store $2^n$ joint probabilities at each iteration for each of the possible sequences of 0's and 1's generated from the current sequence. Hence, for the previous simulation examples, I was only able to produce sequences of length 20 before it became computationally infeasible. Since this is a relatively short sequence, I am also unable to study the run lengths to justify whether the method is working in the way that we expect. Therefore, I either need to produce a different method for simulation or provide a way to limit the computational difficulties.

The average run lengths for the samples from top to bottom are 1.79, 2.33 and 1.99. This emphasises that there is either an error with this method, or that we are simply not generating long enough sequences to be able to view the longer run lengths for the de Bruijn process with conditional word probabilities $\{0.1, 0.1, 0.9, 0.9\}$ and $\{0.1, 0.9, 0.1, 0.9\}$. The average run length for the top plot is as expected, but we would expect the average run lengths for the other two de Bruijn processes to be much larger; i.e. closer to 10, as seen in the directional version in Section 3.3.1 (third sequence in Figure 3.4) and Section 4.2.6 (DBP 4 in Table 4.3).

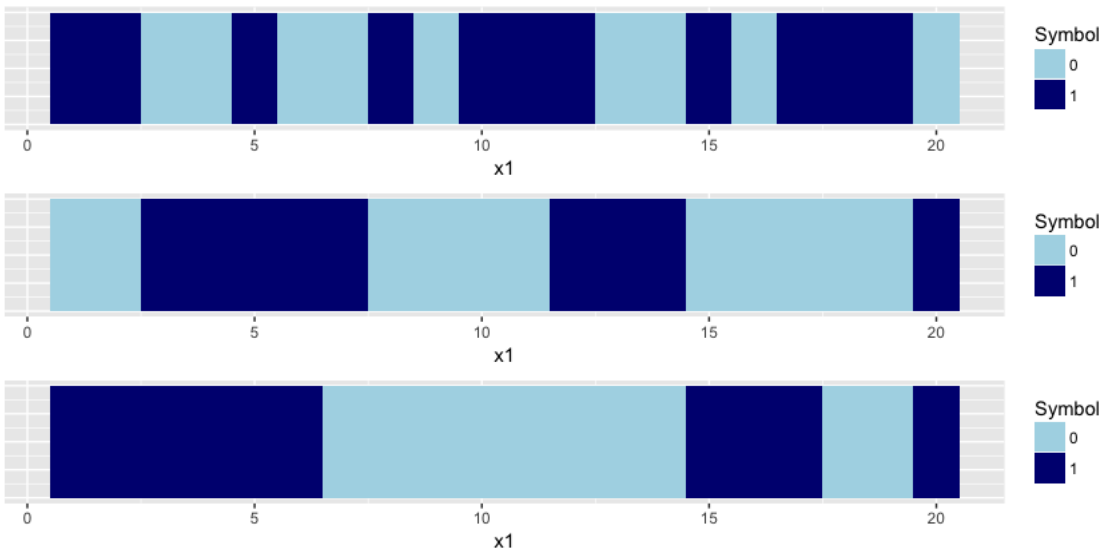



Fig. 6.2 Three samples from length $m = 1$ non-directional de Bruijn processes with letters 0 (light blue) and 1 (dark blue). From top to bottom the conditional word probabilities, $\{p^1_{0:0}, p^1_{0:1}, p^1_{1:0}, p^1_{1:1}\}$, are: $\{0.5, 0.5, 0.5, 0.5\}$, $\{0.1, 0.1, 0.9, 0.9\}$ and $\{0.1, 0.9, 0.1, 0.9\}$.

**Simulation 2**

As an alternative to the above methodology, I attempted to generate a simulation from the de Bruijn process through Gibbs sampling (Brooks et al., 2012; Gilks et al., 1996). I believed this to be possible due to the connection between Markov chains and the de Bruijn words where we can provide the full conditional distributions for the letters. Since I am currently unsure of the Markov nature of the non-directional de Bruijn process, I note that this method may be unsuitable for the given simulation problem.

The purpose of a Gibbs sampler is to generate posterior samples for a given distribution by moving through each variable and sampling from its conditional distribution, given that the remaining variables are fixed at their current values. We would begin with an initial estimate for each of the letters, $v$, in a length $n$ sequence: $(v_1^{(0)}, v_2^{(0)}, ..., v_n^{(0)})$. Then for iteration, $i = 1, 2, ...$, we calculate the following:

$$\begin{aligned} v_1^{(i)} &\sim P(v_1^{(i)}|v_2^{(i-1)}, v_3^{(i-1)}, ..., v_n^{(i-1)}) \\ &\vdots \\ v_j^{(i)} &\sim P(v_j^{(i)}|v_1^{(i)}, ..., v_{j-1}^{(i)}, v_{j+1}^{(i-1)}, ..., v_n^{(i-1)}) \\ &\vdots \\ v_n^{(i)} &\sim P(v_n^{(i)}|v_1^{(i)}, v_2^{(i)}, ..., v_{n-1}^{(i)}), \end{aligned}$$

where,

$$\begin{aligned} v_j^{(i)} &\sim P(v_j^{(i)}|v_1^{(i)}, ..., v_{j-1}^{(i)}, v_{j+1}^{(i-1)}, ..., v_n^{(i-1)}) \\ &= P(v_j^{(i)}|v_{j-m}^{(i)}, ..., v_{j-1}^{(i)}, v_{j+1}^{(i-1)}, ..., v_{j+m}^{(i-1)}) \\ &= p_{w_j}^1, \end{aligned}$$

for a non-directional de Bruijn process with word length $m$, where $p_{w_j}^1$ is the conditional word probability for the letter $v_j$ with associated word, $w_j = (v_{j-m}^{(i)}, ..., v_{j-1}^{(i)}) : (v_{j+1}^{(i-1)}, ..., v_{j+m}^{(i-1)})$. This process continues iteratively until the sequence has converged to a series of 0's and 1's that satisfies the specifications of the de Bruijn process. Providing the sequence of letters, $v_1, v_2, ..., v_n$, remains in the same order, the final

simulation should be consistent irrespective to the ordering of the sampling of each letter in each iteration; i.e., the resulting simulations should have the same properties whether we sample the letters from $v_1$ to $v_n$ or sample the letters from $v_n$ to $v_1$. This is due to the non-directionality of the process.

Four examples are given in Figure 6.3 which are each sequences of length 200 generated with an $m = 1$ word non-directional de Bruijn process. I have again formed each sequence into a loop to avoid any boundary condition complications. The marginal probabilities are kept equal for the first three sequences, with values $\pi(\{0\}) = \pi(\{1\}) = 0.5$, but are changed to $\pi(\{0\}) = 0.2, \pi(\{1\}) = 0.8$ for the final sequence. The set of conditional word probabilities for these examples are, $\{p^1_{0:0}, p^1_{0:1}, p^1_{1:0}, p^1_{1:1}\}$. As with the directional de Bruijn processes, we would expect that letters become far more sticky if both $p^0_{0:0}$ and $p^1_{1:1}$ are chosen to be close to 1. For the examples in Figure 6.3, the conditional word probabilities are set at the following values respectively (top to bottom): $\{0.5, 0.5, 0.5, 0.5\}$, $\{0.1, 0.1, 0.9, 0.9\}$, $\{0.1, 0.9, 0.1, 0.9\}$ and $\{0.775, 0.023, 0.994, 0.9\}$.

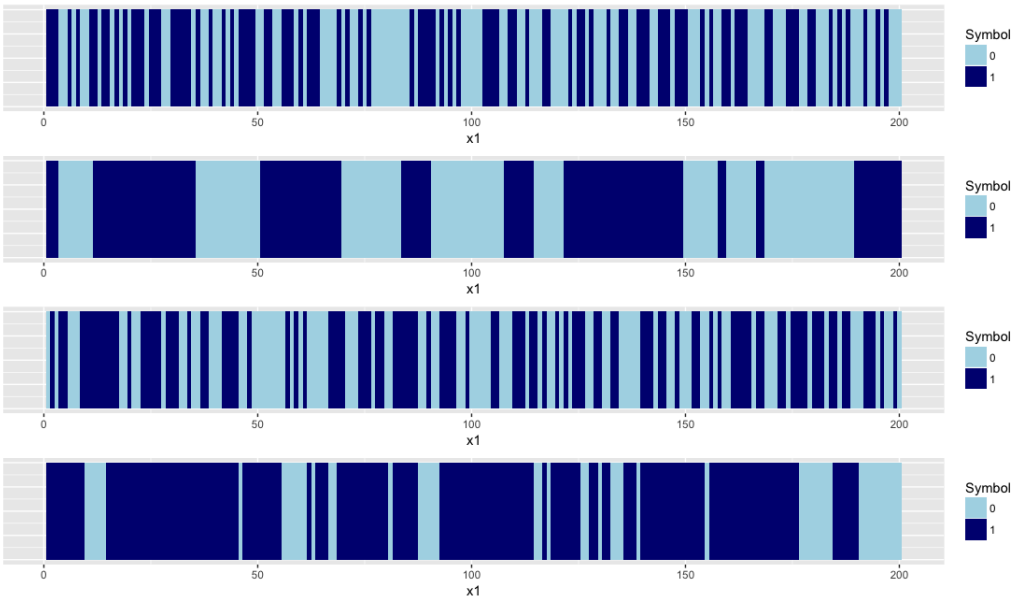



Fig. 6.3 Four samples from length $m = 1$ non-directional de Bruijn processes with letters 0 (light blue) and 1 (dark blue) to show simulation using Gibbs sampling. From top to bottom the conditional word probabilities, $\{p^1_{0:0}, p^1_{0:1}, p^1_{1:0}, p^1_{1:1}\}$, are: $\{0.5, 0.5, 0.5, 0.5\}$, $\{0.1, 0.1, 0.9, 0.9\}$, $\{0.1, 0.9, 0.1, 0.9\}$ and $\{0.775, 0.023, 0.994, 0.9\}$.

The top plot is shown to behave fairly randomly (like random Bernoulli trials), which is as expected, since each conditional word probability is set to be 0.5. The

second and fourth plots are also shown to behave as expected since we are generating much larger run lengths compared to the first plot. The average run lengths of 1's for each of these sequences are 2.04, 11.54 and 8.64 respectively (the first, second and fourth sequences). We can also see that for the fourth sequence (non-symmetric sequence), the total proportion of 1's is much higher than the proportion of 0's at 71%. This is slightly lower than expected since the example was set up with marginal probabilities $\pi(\{1\}) = 0.8$ and $\pi(\{0\}) = 0.2$.

The most surprising result comes from the third sequence, where although $p^0_{0:0} = p^1_{1:1} = 0.9$, the 0's and 1's are occurring far more at random than the second plot. The average run length of 1's for this plot is 1.78. Due to symmetry and the non-directionality that we have forced (since $p^1_{0:1} = 0.1$ and $p^1_{1:0} = 0.9$ for sequence two, but $p^1_{0:1} = 0.9$ and $p^1_{1:0} = 0.1$ for sequence three), we would expect the second and third sequences to have the same run lengths of 0's and 1's. However, we are producing different run lengths depending on the direction of simulation. Therefore, I conclude that there is something wrong with this method, and leave the simulation problem for future work. As stated above, this is likely to be due to not knowing whether the process is Markov or not. I also note here that there are certain constraints that we must make when choosing values for both of these conditional word probabilities ($p^1_{0:1}$ and $p^1_{1:0}$). This is discussed further in Section 6.2.2.

Due to the similarities between this method and the 1-d direction de Bruijn process, I am able to look at the run length distribution. This is described in the following section.

### 6.2.2 Run Length Distribution

The run length distribution for 1-d directional de Bruijn processes was described in Chapter 4. This is a property of great interest, as it is a way to evaluate the stickiness of a particular de Bruijn process. Further, it allows us to assess whether these run lengths are affected by different word lengths and/or transition probabilities. Due to the similarities between the proposed non-directional de Bruijn process and the

directional version explained in Chapters 3 and 4, it is possible to now consider the run lengths of 1's (or 0's) of the non-directional de Bruijn sequences.

As before, a run of 1's is defined by the number of consecutive 1's in a row, which is bounded by a 0 at each end. Very sticky de Bruijn processes are far more likely to produce very long runs, whilst those that are closer to Bernoulli trials are expected to have runs closer to length one. De Bruijn processes that are very anti-sticky will tend to produce very short runs close to one, as they are designed to constantly swap between letters. As a reminder, $p^1_{i:j}$, is the conditional word probability of obtaining a 1 given the associated word is $i:j$.

The run length distribution for a word length $m = 1$ non-directional de Bruijn graph is given in Lemma 6.1. The conditional word probabilities for this process are $\{p^1_{0:0}, p^1_{0:1}, p^1_{1:0}, p^1_{1:1}\}$. For a run length of $n = 1$, we simply require the sequence 010, where this is just defined by the conditional word probability, $p^1_{0:0}$.

For run lengths of two or more, the run sequence must start with the letters 011 and end with the letters 110. These sequences have corresponding probabilities $p^1_{0:1}$ and $p^1_{1:0}$ respectively. This result is similar to the run length distribution from Lemma 4.1 with the exception that because the de Bruijn process is now non-directional, we must have a run-in period at both ends of the sequence of 1's. Due to this non-directionality, we could equally consider the run as going from right to left since the distribution is independent of the direction. $p^1_{0:1} \times p^1_{1:0}$ gives the probability of a run of length $n = 2$, but for a run length of $n \geq 3$, we must include the probability $p^1_{1:1}$. Similarly to Lemma 4.1, we require $p^1_{1:1}$ to be raised to the power $n-2$ since two 1's are generated in the two end run-in probabilities.

**Lemma 6.1** (Run Length Distribution, $m = 1$)**.**

$$P(\textbf{run length} = n) = \begin{cases} p^1_{0:0} & \textit{for } n = 1 \\ p^1_{0:1}(p^1_{1:1})^{n-2}p^1_{1:0} & \textit{for } n \geq 2, \end{cases}$$

*Proof.* $P(\textbf{run length} = n) = p^1_{0:0}$ for $n = 1$. For $n \geq 2$, the run is bounded by the word $0:1$ with probability $p^1_{0:1}$ and the word $1:0$ with probability $p^1_{1:0}$. This generates the sequences 011 and 110 respectively. The run is formed by increasing

the number of 1's with probability $p^1_{1:1}$. For a run of length $n$, this probability is required $n-2$ times, hence: $P(\textbf{run length} = n) = p^1_{0:1}(p^1_{1:1})^{n-2}p^1_{1:0}$, for $n \geq 2$. □

For the simulated examples presented in Section 6.2.1, the choice of conditional word probabilities was very important, and, due to specific constraints of the process, it is not possible to choose the parameters arbitrarily. The four parameters that are in the $m = 1$ examples in Section 6.2.1 are $\{p^1_{0:0}, p^1_{0:1}, p^1_{1:0}, p^1_{1:1}\}$. When initially considering the values that these parameters could take, I assumed we must have $p^1_{0:1} = p^1_{1:0}$ as we were now dealing with non-directional de Bruijn processes; i.e., I did not want to enforce any different structure moving left to right compared with right to left. However, when considering that the distribution in Lemma 6.1 must sum to one for all run length $n$, there is a slightly different constraint that we must obey.

Given that the distribution must sum to 1 over all values of $n$, we have the following:

$$\begin{aligned}
\sum_{n=1}^{\infty} P(\textbf{run length} = n) &= p^1_{0:0} + \sum_{n=2}^{\infty} p^1_{0:1}(p^1_{1:1})^{n-2}p^1_{1:0} \\
&= p^1_{0:0} + p^1_{0:1}p^1_{1:0}\sum_{n=0}^{\infty}(p^1_{1:1})^n \\
&= p^1_{0:0} + \frac{p^1_{0:1}p^1_{1:0}}{1-p^1_{1:1}} \\
&= p^1_{0:0} + \frac{p^1_{0:1}p^1_{1:0}}{p^0_{1:1}}
\end{aligned}$$

since $\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$ when $|x| < 1$. Hence, we can state that:

$$\begin{aligned}
p^1_{0:0} + \frac{p^1_{0:1}p^1_{1:0}}{1-p^1_{1:1}} &= 1 \\
\Rightarrow p^1_{0:0}(1-p^1_{1:1}) + p^1_{0:1}p^1_{1:0} &= 1 - p^1_{1:1} \\
\Rightarrow p^1_{0:0} + p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1} &= 1 - p^1_{1:1} \\
\Rightarrow p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1} &= p^0_{0:0} - p^1_{1:1},
\end{aligned} \tag{6.1}$$

since $p^0_{0:0} + p^1_{0:0} = 1$.

We have always made the assumption that we know the marginal distributions for the letters ($\pi(\{0\})$ and $\pi(\{1\})$). For example, if we knew that there were expected

to be equal numbers of 0's and 1's then we know that $\pi(\{0\}) = \pi(\{1\}) = 0.5$ and $p^0_{0:0} = p^1_{1:1}$. Consequently we then have the following relationship that must hold when defining the conditional word probabilities for the $m = 1$ case:

$$p^1_{0:1}p^1_{1:0} = p^1_{0:0}p^1_{1:1}. \tag{6.2}$$

This can be seen in the examples in Figure 6.2 and in the first three examples in Figure 6.3. For the top sequence in both figures, all parameters are set to be 0.5, and so this satisfies Equation (6.2). For the other two sequences (the second and third sequences of Figure 6.3), we have $p^0_{0:0} = 0.1$, $p^1_{1:1} = 0.9$, and either $p^1_{0:1} = 0.1$, $p^1_{1:0} = 0.9$ (second sequence) or $p^1_{0:1} = 0.9$, $p^1_{1:0} = 0.1$ (third sequence). From Equation (6.2), we must have $p^1_{0:1}p^1_{1:0} = 0.09$ which is satisfied by both of these examples.

Alternatively, if we know that the marginal probabilities for the letters are not equal, then we can state that the parameters, $p^1_{0:0}$ and $p^1_{1:1}$ have the following relationship: $\pi(\{0\})p^1_{1:1} = (1 - \pi(\{0\}))p^0_{0:0} = \pi(\{1\})p^0_{0:0}$. Therefore, equation (6.1) becomes:

$$\begin{aligned}
p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1} &= p^0_{0:0} - p^1_{1:1} \\
\Rightarrow p^1_{0:1}p^1_{1:0} &= p^1_{0:0}p^1_{1:1} - p^1_{1:1} + p^1_{0:0}p^1_{1:1} \\
\Rightarrow p^1_{0:1}p^1_{1:0} &= \left(1 - p^0_{0:0}\right)p^1_{1:1} + p^0_{0:0} - p^1_{1:1} \\
\Rightarrow p^1_{0:1}p^1_{1:0} &= p^1_{1:1} - p^0_{0:0}p^1_{1:1} + p^0_{0:0} - p^1_{1:1} \\
\Rightarrow p^1_{0:1}p^1_{1:0} &= p^0_{0:0}\left(1 - p^1_{1:1}\right) \\
\Rightarrow p^1_{0:1}p^1_{1:0} &= p^0_{0:0}\left(1 - \frac{\pi(\{1\})}{\pi(\{0\})}p^0_{0:0}\right) \\
\Rightarrow p^1_{0:1}p^1_{1:0} &= \alpha
\end{aligned} \tag{6.3}$$

OR

$$\begin{aligned} p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1} &= p^0_{0:0} - p^1_{1:1} \\ \Rightarrow p^1_{0:1}p^1_{1:0} &= p^0_{0:0}\left(1 - p^1_{1:1}\right) \\ \Rightarrow p^1_{0:1}p^1_{1:0} &= \frac{\pi(\{0\})}{\pi(\{1\})}p^1_{1:1}\left(1 - p^1_{1:1}\right) \\ \Rightarrow p^1_{0:1}p^1_{1:0} &= \beta \end{aligned} \tag{6.4}$$

where $\alpha = p^0_{0:0}\left(1 - \frac{\pi(\{1\})}{\pi(\{0\})}p^0_{0:0}\right)$ and $\beta = \frac{\pi(\{0\})}{\pi(\{1\})}p^1_{1:1}\left(1 - p^1_{1:1}\right)$. Parameters must satisfy these constraints to be valid. These constraints are used to define the conditional word probabilities for the de Bruijn process in the last sequence of Figure 6.3.

Specifically, in that example, I set the marginal probabilities to be $\pi(\{0\}) = 0.2$ and $\pi(\{1\}) = 0.8$. Following on from this, I required the sequence to again be sticky for 1's and set $p^1_{1:1} = 0.9$. Given that $\pi(\{0\})p^1_{1:1} = \pi(\{1\})p^0_{0:0}$, this gives $p^0_{0:0} = 0.225$. From (6.3) (or equally (6.4)), we can therefore state,

$$\begin{aligned} p^1_{0:1}p^1_{1:0} &= \alpha \\ &= p^0_{0:0}\left(1 - \frac{\pi(\{1\})}{\pi(\{0\})}p^0_{0:0}\right) \\ &= 0.225 \times \left(1 - \frac{0.8}{0.2} \times 0.225\right) \\ &= 0.0225. \end{aligned} \tag{6.5}$$

Further, given that we also have the marginal constraint, we can state that $\pi(\{0\})p^1_{0:1} = \pi(\{1\})p^0_{1:0}$. Both this result, and that in Equation (6.5), then give the following values for the conditional word probabilities, $\{p^1_{0:0}, p^1_{0:1}, p^1_{1:0}, p^1_{1:1}\} = \{0.775, 0.023, 0.994, 0.9\}$.

The non-directional run length distribution is extended to higher word lengths, $m \geq 2$ in Theorem 6.2, following a similar pattern to both the $m = 1$ non-directional run length distribution and the $m \geq 3$ directional run length distribution.

If we first consider a run length of just $n = 1$, then the distribution will consist of a single probability, as we only require the probability of getting that single letter given the associated de Bruijn word. These words will be of the form $*0 : 0*'$, where both $*$ and $*'$ can be any sequence of letters of length $m - 1$. In terms of the

conditional word probability, this will be $p^1_{*0:0*'}$, where the resulting sequence, to give a run of length one, will be of the form $*010*'$. Due to the non-directionality, we require the run-in period that we had in the directional version at both ends of the run sequence. Hence, there are $2^{2m}$ possible words that take this form. For example, consider a length $m = 2$ non-directional de Bruijn process where we are looking for run lengths $n = 1$. All possible words that take this form are 00100, 00101, 10100 and 10101, which would have the corresponding conditional word probabilities, $p^1_{00:00}$, $p^1_{00:01}$, $p^1_{10:00}$ and $p^1_{10:01}$.

If we now consider run lengths of length $n = 2$, we require two conditional word probabilities for each of the 1's in the run. The first of these probabilities, $p^1_{*0:10*'}$, refers to a word of the form $*0 : 10*'$ where the sequence represented by $*$ is of length $m - 1$. Given that we know there will be exactly two 1's, the second half of the word must be of the form $10*'$, where the sequence represented by $*'$ is now of length $m - 2$. The second conditional word probability, $p^1_{*01:0*'}$, refers to a word of the form $*01 : 0*'$. This is a mirror image of the first word, since, due to the form of non-directional words, we require a run-in period at both ends of the run of 1's. $*$ is now any sequence of length $m - 2$ and $*'$ is any sequence of length $m - 1$. The resulting sequence is of the form $*0110*'$, where both $*$ and $*'$ are of length $m - 1$. There are $2^{2m}$ possible ways to form this sequence.

For run lengths longer than $n = 2$, the structure is very similar to above, where we require the same run-in conditional word probabilities at both ends of the run of 1's. If the run length is larger than the word length, $m$, then the first words on both ends will be of the form $*0 : 1...1$ and $1...1 : 0*'$. Both $*$ and $*'$ are any possible sequences of 0's and 1's of length $m - 1$. Moving from the outer words to the inner words, we decrease the length of the sequences $*$ and $*'$, and include the next letter 1 into this part of the word; i.e. the word $*0 : 1...1$ goes to the word $*01 : 1...1$ (where $*$ is now of length $m - 2$), then to the word $*011 : 1...1$, carrying on in this way until we reach the word $01...1 : 1...1$. This pattern occurs on both ends of the run-up. For runs of length $n > 2m + 1$, the probability $p^1_{1...1:1...1}$ is included $n - 2m$ times since $m$ 1's are included in the run-in periods on both sides of the run.

For run lengths between $n = 2$ and $n = 2m$, the length of the word is longer than the run itself, and we thus never reach the point where we require the probability $p^1_{1\ldots1:1\ldots1}$. The run-in periods at each end will have to accommodate this by also including letters from the opposite end for certain words.

Since there are different possibilities for both the start sequence and the end sequence which are included in the de Bruijn words, we must again use the law of total probability in the same way to Theorem 4.2. The equation is used as follows:

$$P(\textbf{run length} = n) = \sum_{i=0}^{2^{2(m-1)}-1} P(\textbf{run length} = n | * \_ *'_i)\pi(*\_*'_i), \tag{6.6}$$

where the notation, $*\_*'_i$, is used to represent the $i^{\text{th}}$ combination of both of the length $m-1$ sequences which are contained in the end words of a non-directional run length. Hence, $\pi(*\_*'_i)$ is the $i^{\text{th}}$ joint marginal probability for a run to start with the sequence $*$ and end with the sequence $*'$. Since the start sequence must be of the form $*0$ and the end sequence is of the form $0*'$, where both $*$ and $*'$ are sequences of length $m-1$, there are $2^{2(m-1)}$ possible combinations that could contain the same run of 1's.

In Chapter 4, the joint marginal probabilities in Equation (6.6) were found by applying the law of total probability again, so that the marginal probabilities of the $m-2$ sequences were found in terms of the marginal probabilities of the length $m$ de Bruijn words. The marginal probabilities of the de Bruijn words are equivalent to the stationary distribution of the de Bruijn process Markov chain, which was found by either powering up the transition matrix or by calculating the corresponding eigenvectors. Unfortunately, we can no longer use this process to find the marginal probabilities of certain sequences for the non-directional de Bruijn process. The main problem here is that we require calculation of the joint marginal probability of simultaneously getting two length $m-1$ sequences (at the beginning of the run and at the end of the run), given that we are working with a length $m$ de Bruijn process. We also do not have a transition matrix, and so cannot calculate the stationary distributions of the words in the de Bruijn process. Therefore, I leave this part of the run length distribution for future work. Currently, I plan to use a recurrence

relationship for the marginal probabilities of length $m-1$ de Bruijn words in terms of length $m$ de Bruijn words. If we have a given sequence from the de Bruijn process, then it may be possible to estimate the marginal probabilities ($\pi(*\_*'_i)$) by counting the number of times each combination of start ($*$) and end ($*'$) sequences occur around a run of 1's.

**Theorem 6.2** (Run Length Distribution, $m \geq 2$)**.**

$$
P(\textbf{run length} = n)
$$

$$
= \begin{cases}
\sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} & \textit{for } n = 1 \\
\sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) & \textit{for } n = 2:2m \\
\quad \times \prod_{k=0}^{n-1} \left\{ p^1_{2^{min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] : \right. & \\
\quad \left. \sum_{t=0}^{min(n-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{n-k-1}} \left[j - (j \bmod 2^{n-k-1})\right] \right\} & \\
\sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) & \textit{for } n \geq 2m+1 \\
\quad \times \prod_{k=0}^{2m-1} \left\{ p^1_{2^{min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] : \right. & \\
\quad \left. \sum_{t=0}^{min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[j - (j \bmod 2^{2m-k-1})\right] \right\} & \\
\quad \times \left( p^1_{2^m-1:2^m-1} \right)^{n-2m} &
\end{cases}
$$

*Proof.* See Appendix B □

For example, assume that we have a non-directional de Bruijn process of word length $m = 2$. We thus require words of the form $*0:11$ and $11:0*'$, at either ends of the run of 1's, where both $*$ and $*'$ can either be a 0 or a 1 to complete each of these words. Hence for $*0:11$, we can either have the word $00:11$ or $10:11$, and for $11:0*'$, we can either have the word $11:00$ or $11:01$. Obtaining a 1 in the place of : for all of these words gives the sequences 00111, 10111, 11100 and 11101 respectively.

For the next word in (moving towards the run of 1's) on either end of the run, we no longer have the sequences $*$ and $*'$, and so we have the word $01:11$ in the left

run-in and the word $11:10$ in the right run-in. These will be the next words in the run regardless of the words on the ends.

Finally, we require the word $11:11$ until the desired length of 1's is reached. Using Equation (6.6), we hence have the following for the probability of a run length of size $n$:

$$\begin{aligned} P(\textbf{run length} = n) = \pi(0\_0)&\left[p^1_{00:11}p^1_{01:11}\left(p^1_{11:11}\right)^{n-2}p^1_{11:10}p^1_{11:00}\right] \\ + \pi(0\_1)&\left[p^1_{00:11}p^1_{01:11}\left(p^1_{11:11}\right)^{n-2}p^1_{11:10}p^1_{11:01}\right] \\ + \pi(1\_0)&\left[p^1_{10:11}p^1_{01:11}\left(p^1_{11:11}\right)^{n-2}p^1_{11:10}p^1_{11:00}\right] \\ + \pi(1\_1)&\left[p^1_{10:11}p^1_{01:11}\left(p^1_{11:11}\right)^{n-2}p^1_{11:10}p^1_{11:01}\right], \end{aligned}$$

where $\pi(0\_1)$ is the joint marginal probability of obtaining a 0 and a 1 in place of $*$ and $*'$ in the two words at the end of the run, $*0:11$ and $11:0*'$, respectively.

### 6.2.3 Expected Run Length

Given the non-directional run length distribution, we can now calculate the expectation. The expected run length for an $m = 1$ non-directional de Bruijn process is given in Lemma 6.3. Unlike the results of Section 4.2.2, we can no longer utilise connections to the geometric distribution to help refine the results. In Lemma 4.3, we obtained an expression in the form $\sum_n n(1-p)^n p$, which we were able to simplify to $\frac{1-p}{p}$ using the definition for the expectation of a geometric distribution. In Lemma 6.3 we are instead left with the expression $\sum_{n=2}^{\infty} n\left(p^1_{1:1}\right)^n p^1_{1:0}$. Although this looks similar to the expression in Lemma 4.3, it may not be the case that $p^1_{1:1} + p^1_{1:0} = 1$, meaning that we can no longer use the geometric distribution. Instead, we can use the relationship, $\sum_{n=0}^{\infty} np^n = \frac{p}{(p-1)^2}$ to simplify results, providing $|p| < 1$. Note that we are disregarding any cases with $p = 1$, as the de Bruijn process would produce a sequence consisting entirely of 1's (or 0's).

**Lemma 6.3** (Expected Run Length, $m = 1$)**.**

$$E\left[\textbf{run length}\right] = p^1_{0:0} + \frac{p^1_{0:1}p^1_{1:0}\left(2 - p^1_{1:1}\right)}{\left(1 - p^1_{1:1}\right)^2} \qquad \text{for } \mathrm{p}^1_{1:1} < 1$$

*Proof.*

$$
\begin{aligned}
E\left[\textbf{run length}\right] &= \sum_{n=1}^{\infty} n \times P(\textbf{run length} = n) \\
&= p^1_{0:0} + \sum_{n=2}^{\infty} n p^1_{0:1} \left(p^1_{1:1}\right)^{n-2} p^1_{1:0} \\
&= p^1_{0:0} + \frac{p^1_{0:1} p^1_{1:0}}{\left(p^1_{1:1}\right)^2} \sum_{n=2}^{\infty} n \left(p^1_{1:1}\right)^n \\
&= p^1_{0:0} + \frac{p^1_{0:1} p^1_{1:0}}{\left(p^1_{1:1}\right)^2} \left[\sum_{n=0}^{\infty} n \left(p^1_{1:1}\right)^n - p^1_{1:1}\right] \\
&= p^1_{0:0} + \frac{p^1_{0:1} p^1_{1:0}}{\left(p^1_{1:1}\right)^2} \left[\frac{p^1_{1:1}}{\left(1 - p^1_{1:1}\right)^2} - p^1_{1:1}\right] \qquad \text{since } p^1_{1:1} < 1 \\
&= p^1_{0:0} + \frac{p^1_{0:1} p^1_{1:0} \left(2 - p^1_{1:1}\right)}{\left(1 - p^1_{1:1}\right)^2}.
\end{aligned}
$$

□

The expected run length of non-directional de Bruijn processes with word length two or higher is given in Theorem 6.4. The result $\sum_{n=0}^{\infty} np^n = \frac{p}{(p-1)^2}$ can still be used to simplify results, providing $|p| < 1$.

**Theorem 6.4** (Expected Run Length, $m \geq 2$)**.**

$$
\begin{aligned}
&E\left[\textbf{run length}\right] \\
&= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
&+ \sum_{l=2}^{2m} l \left[ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{l-1} \left\{ p^1_{2^{min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] : \right.\right. \\
&\qquad \left.\left. \sum_{t=0}^{min(l-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{l-k-1}} \left[j - (j \bmod 2^{l-k-1})\right] \right\} \right] \\
&+ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{2m-1} \left\{ p^1_{2^{min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] : \right. \\
&\qquad \left. \sum_{t=0}^{min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[j - (j \bmod 2^{2m-k-1})\right] \right\} \\
&\quad \times \left\{ \left( \frac{1}{p^1_{2^m-1:2^m-1}} \right)^{2m} \left[ \frac{p^1_{2^m-1:2^m-1}}{(1 - p^1_{2^m-1:2^m-1})^2} - \sum_{s=0}^{2m} s (p^1_{2^m-1:2^m-1})^s \right] \right\}
\end{aligned}
$$

*Proof.* See Appendix B □

### 6.2.4 Variance of Run Length

The next measure of interest is the variance of the run length distribution. We can calculate the variance using the results from Lemma 6.1 and Theorem 6.2, along with the well known definition, Var[**run length**] = $E$[**run length**$^2$] − ($E$[**run length**])$^2$. The squared expectation of the run length distribution for the $m = 1$ word length case is given in Lemma 6.5. Again, it is no longer possible to use properties from the geometric distribution to help make simplifications to results. In Lemma 6.5, we are left with the expression, $\sum_{n=2}^{\infty} n^2 \left(p^1_{1:1}\right)^n$, which we can simplify using the relationship, $\sum_{n=0}^{\infty} n^2 p^n = -\frac{p(p+1)}{(p-1)^3}$ providing $|p| < 1$.

**Lemma 6.5** (Squared Expectation of Run Length, $m = 1$)**.**

$$E\left[\textbf{run length}^2\right] = p^1_{0:0} - \frac{p^1_{0:1}p^1_{1:0}\left(\left(p^1_{1:1}\right)^2 - 3p^1_{1:1} + 4\right)}{\left(p^1_{1:1} - 1\right)^3} \qquad \text{for } \mathrm{p}^1_{1:1} < 1$$

*Proof.*

$$\begin{aligned}
E\left[\textbf{run length}^2\right] &= \sum_{n=1}^{\infty} n^2 \times P(\textbf{run length} = n) \\
&= p^1_{0:0} + \sum_{n=2}^{\infty} n^2 p^1_{0:1}\left(p^1_{1:1}\right)^{n-2} p^1_{1:0} \\
&= p^1_{0:0} + \frac{p^1_{0:1}p^1_{1:0}}{\left(p^1_{1:1}\right)^2} \sum_{n=2}^{\infty} n^2 \left(p^1_{1:1}\right)^n \\
&= p^1_{0:0} + \frac{p^1_{0:1}p^1_{1:0}}{\left(p^1_{1:1}\right)^2}\left[\sum_{n=0}^{\infty} n^2 \left(p^1_{1:1}\right)^n - p^1_{1:1}\right] \\
&= p^1_{0:0} - \frac{p^1_{0:1}p^1_{1:0}}{\left(p^1_{1:1}\right)^2}\left[\frac{p^1_{1:1}(p^1_{1:1}+1)}{\left(p^1_{1:1}-1\right)^3} + p^1_{1:1}\right] \qquad \text{since } p^1_{1:1} < 1 \\
&= p^1_{0:0} - \frac{p^1_{0:1}p^1_{1:0}\left(\left(p^1_{1:1}\right)^2 - 3p^1_{1:1} + 4\right)}{\left(p^1_{1:1} - 1\right)^3}.
\end{aligned}$$

□

The squared expectation of the run length distribution for word lengths $m \geq 2$ is given in Theorem 6.6. Again, the variance is found by substituting this result into the known result, Var[**run length**] = $E$[**run length**$^2$] − ($E$[**run length**])$^2$. $E$[**run length**] is given in Theorem 6.4 and the relationship, $\sum_{n=0}^{\infty} n^2 p^n = -\frac{p(p+1)}{(p-1)^3}$ is used providing $|p| < 1$.

**Theorem 6.6** (Squared Expectation of Run Length, $m \geq 2$)**.**

$$
\begin{aligned}
&E\left[\textbf{run length}^2\right] \\
&= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
&+ \sum_{l=2}^{2m} l^2 \left[ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{l-1} \left\{ p^1_{2^{min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right.\right. \\
&\qquad \left.\left. \sum_{t=0}^{min(l-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{l-k-1}} \left[ j - (j \bmod 2^{l-k-1}) \right] \right\} \right] \\
&+ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{2m-1} \left\{ p^1_{2^{min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right. \\
&\qquad \left. \sum_{t=0}^{min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j - (j \bmod 2^{2m-k-1}) \right] \right\} \\
&\quad \times \left\{ \left( \frac{1}{p^1_{2^m-1:2^m-1}} \right)^{2m} \left[ \frac{p^1_{2^m-1:2^m-1}(p^1_{2^m-1:2^m-1}+1)}{(p^1_{2^m-1:2^m-1}-1)^3} + \sum_{s=0}^{2m} s^2 (p^1_{2^m-1:2^m-1})^s \right] \right\}
\end{aligned}
$$

for $\mathrm{p}^1_{2^{\mathrm{m}}-1:2^{\mathrm{m}}-1} < 1$

*Proof.* See Appendix B □

### 6.2.5 Examples

Having calculated the run length distribution, we can now discuss the analytical run lengths for the examples in Section 6.2.1. In Figures 6.2 and 6.3, we had four examples of length $m = 1$ non-directional de Bruijn processes with the conditional word probabilities $\{p^1_{0:0}, p^1_{0:1}, p^1_{1:0}, p^1_{1:1}\}$ being $\{0.5, 0.5, 0.5, 0.5\}$, $\{0.1, 0.1, 0.9, 0.9\}$, $\{0.1, 0.9, 0.1, 0.9\}$ and $\{0.775, 0.023, 0.994, 0.9\}$.

Table 6.1 shows the probabilities of getting a run length of $n$ 1's using the $m = 1$ word run length distribution in Lemma 6.1 for these four examples (presented in the same order as above). We immediately notice that the expected run lengths for the de Bruijn processes with conditional word probabilities $\{0.1, 0.1, 0.9, 0.9\}$ and $\{0.1, 0.9, 0.1, 0.9\}$ are now equal. This is what we expected, and further proves that there are errors in both of the simulation methods presented in Section 6.2.1. The results are similar to that seen in Table 4.1, for the 1-d directional de Bruijn processes. Both DBP 1 (Bernoulli trials) and DBP 4 (non-symmetrical process) are shown to have a high chance of a short run length, where the probabilities of getting

a run of just $n = 1$ are 50% and 77.5% respectively. The other two stickier processes have a much lower chance of a short run length, and are also shown to stay fairly consistent for similar run lengths.

| Run Length, n | DBP 1 | DBP 2 | DBP 3 | DBP 4 |
|---|---|---|---|---|
| 1 | 0.5 | 0.1 | 0.1 | 0.775 |
| 2 | 0.25 | 0.09 | 0.09 | 0.0225 |
| 3 | 0.125 | 0.081 | 0.081 | 0.0203 |
| 4 | 0.0625 | 0.0729 | 0.0729 | 0.0182 |
| 5 | 0.0313 | 0.0656 | 0.0656 | 0.0164 |
| 6 | 0.0156 | 0.0590 | 0.0590 | 0.0148 |
| 7 | 0.00781 | 0.0531 | 0.0531 | 0.0133 |
| 8 | 0.00391 | 0.0478 | 0.0478 | 0.0120 |

Table 6.1 Table to show the probabilities of getting run lengths of $n = 1, ..., 10$ for four different non-directional de Bruijn processes of word length $m = 1$. The corresponding conditional word probabilities ($\{p^1_{0:0}, p^1_{0:1}, p^1_{1:0}, p^1_{1:1}\}$) for these four processes are as follows, DBP 1: $\{0.5, 0.5, 0.5, 0.5\}$, DBP 2: $\{0.1, 0.1, 0.9, 0.9\}$ , DBP 3: $\{0.1, 0.9, 0.1, 0.9\}$, DBP 4: $\{0.775, 0.023, 0.994, 0.9\}$.

The theoretical expected run lengths of 1's for these four sequences (DBP 1, DBP 2, DBP 3 and DBP 4) are 2, 10, 10 and 3.25 respectively. These values are as expected, as DBP 2 and DBP 3 are designed to produce sticky sequences of 0's and 1's, whilst DBP 1 is designed to produce random sequences. DBP 4 is interesting since, although we have set the conditional word probabilities to produce sticky sequences, the expected run length is fairly short at 3.25. This is likely to be due to the intermediate conditional word probabilities, $p^1_{0:1}$ and $p^1_{1:0}$. Setting $p^1_{0:1} = 0.023$ causes there to be a low chance of observing a run of 1's longer than one. This is confirmed in Table 6.1, where we see that the probability of a run of length $n = 1$ is large at 77.5%. With the results from Table 6.1 we can conclude that for this non-symmetrical process, it is very hard to start a run of 1's longer than one, but once we have started the run off, we have equal chances of observing longer runs. Therefore, by using all available conditional word probabilities, we can create sequences with a vast amount of structure.

Finally, the theoretical variances for the four de Bruijn processes are 6, 190, 190 and 48.25 showing that there is more variability in the stickier processes.

### 6.2.6 Generating Functions

#### Probability Generating Function

We can also find generating functions (Johnson et al., 2005; Wilf, 1994) for the non-directional de Bruijn process run length distribution. The first of these is the probability generating function, which takes the form $G(x) = E[x^n] = \sum_{n=1}^{\infty} x^n P(\textbf{run length} = n)$. As a reminder, the probability of a run of length $s$ is found by evaluating the $s^{\text{th}}$ derivative of $G$ at $x = 0$, and dividing by $s!$ so that $P(X = s) = \frac{G^{(s)}(0)}{s!}$. See Section 4.2.4 and Johnson et al. (2005); Wilf (1994) for more information and background reading on generating functions

The probability generating function for the de Bruijn process with word length $m = 1$ is given in Lemma 6.7. The result is very similar to that given for the directional version with word length $m = 2$ in Lemma 4.10. Unlike the expectation and variance of the non-directional run length distribution, we are now able to follow the same format as the generating function for the geometric distribution seen in Theorem 4.9.

**Lemma 6.7** (Run Length Probability Generating Function, $m = 1$)**.**

$$G(x) = \frac{(p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1})\,x^2 + p^1_{0:0}x}{1 - p^1_{1:1}x}$$

*Proof.*

$$P(\textbf{run length} = n) = \begin{cases} p^1_{0:0} & \text{for } n = 1 \\ p^1_{0:1}(p^1_{1:1})^{n-2}p^1_{1:0} & \text{for } n \geq 2, \end{cases}$$

The recurrence relationship has the following form:

$$\begin{aligned} a_0 &= 0 \\ a_1 &= p^1_{0:0} \\ a_2 &= p^1_{0:1}p^1_{1:0} \\ a_{n+1} &= p^1_{1:1}a_n \qquad \text{for } n \geq 2 \end{aligned}$$

This is solved to find the generating function $G(x) = \sum_{n\geq 0} a_n x^n$. Then, multiplying by $x^n$ and summing over $n$ gives the following:

$$\begin{aligned}
\sum_{n\geq 2} a_{n+1}x^n &= \sum_{n\geq 2} p^1_{1:1} a_n x^n \\
\left(a_3x^2 + a_4x^3 + a_5x^4 + \ldots\right) &= p^1_{1:1}\left(\sum_{n\geq 0} a_n x^n - a_0 - a_1 x\right) \\
\frac{1}{x}\left[\left(a_0 + a_1x + a_2x^2 + \ldots\right) - a_0 - a_1x - a_2x^2\right] &= p^1_{1:1}\left(G(x) - p^1_{0:0}x\right) \\
\frac{G(x) - p^1_{0:0}x - p^1_{0:1}p^1_{1:0}x^2}{x} &= p^1_{1:1}\left(G(x) - p^1_{0:0}x\right) \\
G(x) &= \frac{\left(p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1}\right)x^2 + p^1_{0:0}x}{1 - p^1_{1:1}x}
\end{aligned}$$

□

The probability generating function for the run length distribution of non-directional de Bruijn processes with word length $m \geq 2$ is given in Theorem 6.8. Again, it has a similar structure to both the $m = 1$ word length case and the $m \geq 3$ directional case in Theorem 4.11. There are still additional polynomial terms which arise from the words during the run-in periods at both ends of the run (before we reach words of the form 1...1 : 1...1). There is no recurrence relationship that can describe the pattern between each of these terms in the generating function, hence it is formulated as a polynomial. For the main part of the run, where we only have words of the form 1...1 : 1...1, this becomes equivalent to a geometric distribution, and hence the result becomes equivalent to the geometric generating function.

**Theorem 6.8** (Run Length Probability Generating Function, $m \geq 2$)**.**

$$G(x) = \sum_{s=0}^{2m} a_s x^s + \frac{p^1_{2^m-1:2^m-1} a_{2m} x^{2m+1}}{1 - p^1_{2^m-1:2^m-1} x}$$

where,

$$a_1 = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j}$$

$$a_2 = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \left\{ p^1_{2i\,:\,2^{m-1}+\frac{1}{2}[j-(j \bmod 2)]} \right\} \times \left\{ p^1_{1+4[i \bmod 2^{m-2}]\,:\,j} \right\}$$

$$\vdots$$

$$a_{2m} = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \times \prod_{k=0}^{2m-1} \left\{ p^1_{2^{min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]\,:\, \sum_{t=0}^{min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}}\left[j-(j \bmod 2^{2m-k-1})\right]} \right\}$$

*Proof.* See Appendix B □

### Moment Generating Function

We can also find the moment generating functions for both the $m = 1$ and $m \geq 2$ non-directional de Bruijn process run length distributions. Moment generating functions take the form $G(x) = E[e^{nx}] = \sum_{n=1}^{\infty} e^{nx} P(\textbf{run length} = n)$ for $n = 0, 1, 2, ...$, and are used to find the moments of the distribution. The $s^{\text{th}}$ moment, can be found by evaluating the $s^{\text{th}}$ derivative of $G$ at $x = 0$ such that, $m_s = G^{(s)}(0)$. $G(0)$ always exists for the generating function and is equal to one. From Section 4.2.4 we can also find the cumulant generating function by taking the log of the moment generating functions given below (McCullagh, 1987; Wilf, 1994).

The moment generating function for the $m = 1$ case is given in Lemma 6.9. There is a clear similarity between this result and the result shown in Lemma 4.13. If we differentiate the moment generating function, we can obtain the expression for the expected run length given in Lemma 6.3 (this calculation is given in Appendix B). Using Maple, we can further obtain the result for the squared expected run length in Lemma 6.5.

**Lemma 6.9** (Run Length Moment Generating Function, $m = 1$)**.**

$$G(x) = \frac{(p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1})\, e^{2x} + p^1_{0:0}e^x}{1 - p^1_{1:1}e^x}$$

*Proof.*

$$P(\textbf{run length} = n) = \begin{cases} p^1_{0:0} & \text{for } n = 1 \\ p^1_{0:1}(p^1_{1:1})^{n-2}p^1_{1:0} & \text{for } n \geq 2, \end{cases}$$

The recurrence relationship has the following form:

$$\begin{aligned} a_0 &= 0 \\ a_1 &= p^1_{0:0} \\ a_2 &= p^1_{0:1}p^1_{1:0} \\ a_{n+1} &= p^1_{1:1}a_n \qquad \text{for } n \geq 2 \end{aligned}$$

This is solved to find the generating function $G(x) = \sum_{n\geq 0} a_n e^{nx}$. Then, multiplying by $e^{nx}$ and summing over $n$ gives the following:

$$\begin{aligned} \sum_{n\geq 2} a_{n+1}e^{nx} &= \sum_{n\geq 2} p^1_{1:1}a_n e^{nx} \\ \left(a_3e^{2x} + a_4e^{3x} + a_5e^{4x} + \ldots\right) &= p^1_{1:1}\left(\sum_{n\geq 0} a_n e^{nx} - a_0 - a_1e^x\right) \\ \frac{1}{e^x}\left[\left(a_0 + a_1e^x + a_2e^{2x} + \ldots\right) - a_0 - a_1e^x - a_2e^{2x}\right] &= p^1_{1:1}\left(G(x) - p^1_{0:0}e^x\right) \\ \frac{G(x) - p^1_{0:0}e^x - p^1_{0:1}p^1_{1:0}e^{2x}}{e^x} &= p^1_{1:1}\left(G(x) - p^1_{0:0}e^x\right) \\ G(x) &= \frac{(p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1})\, e^{2x} + p^1_{0:0}e^x}{1 - p^1_{1:1}e^x} \end{aligned}$$

□

Lastly, the moment generating function for the non-directional run length distribution when $m \geq 2$ is given in Theorem 6.10. This has a similar form to both the $m = 1$ case and the directional case for $m \geq 3$. The additional polynomial

terms are once again present, coming from the end run-in words, where we are either leading into or out of the run of 1's. The simplified fraction term is of the same form as the moment generating function for the geometric distribution. Using Maple, we can differentiate the moment generating function to obtain the expressions for the expected run length and squared expected run length, given in Theorems 6.4 and 6.6.

**Theorem 6.10** (Run Length Moment Generating Function, $m \geq 3$)**.**

$$G(x) = \sum_{s=0}^{2m} a_s e^{sx} + \frac{p^1_{2^m-1:2^m-1} a_{2m} e^{(2m+1)x}}{1 - p^1_{2^m-1:2^m-1} e^x}$$

where,

$$a_1 = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j}$$

$$a_2 = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \left\{ p^1_{2i\,:\,2^{m-1}+\frac{1}{2}[j-(j \bmod 2)]} \right\} \times \left\{ p^1_{1+4[i \bmod 2^{m-2}]\,:\,j} \right\}$$

$$\vdots$$

$$a_{2m} = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \times \prod_{k=0}^{2m-1} \left\{ p^1_{2^{min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]\,:\, \sum_{t=0}^{min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}}\left[j-(j \bmod 2^{2m-k-1})\right]} \right\}$$

*Proof.* See Appendix B □

### 6.2.7 Marginal Likelihood

Given that we can define a one-to-one relationship between the non-directional words and the 1-d directional words, we can extend the directional word marginal likelihood (from Lemma 4.17 and Theorem 4.18) to be applicable to the non-directional de Bruijn process. The marginal likelihood is the joint distribution of a sequence in terms of the proportion of different words in that given sequence. In exactly the same way as Section 4.3, given a sequence of 0's and 1's, we will be able to use the

marginal likelihood to estimate the marginal probabilities for each of the de Bruijn words. This can be done using either maximum likelihood or Bayesian inference.

The marginal likelihood for a de Bruijn process of word length $m = 1$ is given in Lemma 6.11, which is equivalent to counting the proportion of each of the words in the sequence. As a reminder, $\pi(i : j)$, is used to represent the marginal probability for the word $i : j$, and $n_{i:j}$ is the number of times the word $i : j$ appears in the sequence. Dimensionality is reduced slightly since all marginal probabilities sum to one and all the $n$'s add up to the total sum of all the words, defined as $N$.

**Lemma 6.11** (Marginal Likelihood, $m = 1$)**.**

$$\begin{aligned}\mathcal{L}(seq) &= \pi(0:0)^{n_{0:0}}\pi(0:1)^{n_{0:1}}\pi(1:0)^{n_{1:0}}\pi(1:1)^{n_{1:1}} \\ &= \pi(0:0)^{n_{0:0}}\pi(0:1)^{n_{0:1}}\pi(1:0)^{n_{1:0}} \\ &\quad \times [1-(\pi(0:0)+\pi(0:1)+\pi(1:0))]^{N-(n_{0:0}+n_{0:1}+n_{1:0})}\end{aligned}$$

for $N = n_{0:0} + n_{0:1} + n_{1:0} + n_{1:1}$ and $\pi(0:0) + \pi(0:1) + \pi(1:0) + \pi(1:1) = 1$

*Proof.* Given a sequence of letters 0 and 1 of length $N+1$, the sequence can be defined by its associated de Bruijn words of length $m = 1$. $\pi(i : j)$ gives the probability of obtaining the word $i : j$. The joint distribution is given by the product of the respective $\pi$'s for each word and $N$ gives the total number of words. By collecting like terms, the above definition is given. □

The marginal likelihood for a de Bruijn process with general word length $m$ is given in Theorem 6.12, following the same structure as in both the directional version (Theorem 4.18) and the non-directional version for $m = 1$ (Lemma 6.11). As used previously in this thesis, I have chosen to write the result in terms of the decimal representation of the binary values.

**Theorem 6.12** (Marginal Likelihood, $m \geq 1$)**.**

$$\begin{aligned}\mathcal{L} &= \prod_{i=0}^{2^{2m}-1} \pi\left(\frac{1}{2^m}\left[i-(i \bmod 2^m)\right] \,:\, i \bmod 2^m\right)^{n_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}} \\ &= \prod_{i=0}^{2^{2m}-1} \pi(d(i))^{n_{d(i)}} \\ &= \prod_{i=0}^{2^{2m}-2} \pi(d(i))^{n_{d(i)}} \left[1-\left(\sum_{j=0}^{2^{2m}-2} \pi(d(j))\right)\right]^{N-\left(\sum_{j=0}^{2^{2m}-2} n_{d(j)}\right)}\end{aligned}$$

where, $d(i) = \frac{1}{2^m}\left[i-(i \bmod 2^m)\right] \,:\, i \bmod 2^m$

*Proof.* See Appendix B □

### 6.2.8 Conditional Word Likelihood

As with the directional process, we can now define the likelihood for the conditional word probabilities of a sequence generated from a non-directional de Bruijn process. Given the likelihood, we will then be able to estimate the conditional word probabilities using either maximum likelihood or Bayesian inference (since the conditional word probabilities are the non-directional equivalent to the directional transition probabilities). For the same reasons as before, we assume that we know what the word length used to generate the sequence is before we begin estimation, as a different number of conditional word probabilities are needed for each word length.

The likelihood of a sequence in terms of its conditional word probabilities for a length $m = 1$ non-directional de Bruijn process is given in Lemma 6.13. As with Lemma 4.19, the number of times each conditional word probability occurs in a sequence is very important since this defines the exact ordering of the letters for that sequence. The de Bruijn structure ensures that only certain probabilities of obtaining a 1 can occur next to each other, and so gives us the unique ordering of both the words and letters. Therefore, the joint distribution of the sequence in terms of the conditional word probabilities can be described by the product of all the subsequent conditional word probabilities. $p^1_{i:j}$ is the probability of getting a 1 in the sequence $i * j$ and $n^1_{i:j}$ is the number of times the sequence $i1j$ occurs with word $i : j$.

**Lemma 6.13** (Conditional Word Likelihood, $m = 2$)**.**

$$\begin{aligned}\mathcal{L} &= \left(p^0_{0:0}\right)^{n^0_{0:0}} \left(p^1_{0:0}\right)^{n^1_{0:0}} \left(p^0_{0:1}\right)^{n^0_{0:1}} \left(p^1_{0:1}\right)^{n^1_{0:1}} \left(p^0_{1:0}\right)^{n^0_{1:0}} \left(p^1_{1:0}\right)^{n^1_{1:0}} \left(p^0_{1:1}\right)^{n^0_{1:1}} \left(p^1_{1:1}\right)^{n^1_{1:1}} \\ &= \left(1-p^1_{0:0}\right)^{n^0_{0:0}} \left(p^1_{0:0}\right)^{n^1_{0:0}} \left(1-p^1_{0:1}\right)^{n^0_{0:1}} \left(p^1_{0:1}\right)^{n^1_{0:1}} \left(1-p^1_{1:0}\right)^{n^0_{1:0}} \left(p^1_{1:0}\right)^{n^1_{1:0}} \\ &\quad \times \left(1-p^1_{1:1}\right)^{n^0_{1:1}} \left(p^1_{1:1}\right)^{n^1_{1:1}}\end{aligned}$$

*Proof.* Assume a sequence of $n$ letters, $L = l_1, l_2, ..., l_n$, where $l_i \in [0, 1]$ and the ordering is fixed. This sequence can be expressed in terms of its de Bruijn words such that $L = w_{l_1:l_3}, w_{l_2:l_4}, ..., w_{l_{n-2}:l_n}$, where $w_{l_i:l_{i+2}}$ is the de Bruijn word of length $m = 1$ that consists of the letters $l_i$ and $l_{i+2}$. Consider the joint distribution of this sequence. Starting from $w_{l_1:l_3}$, $l_2$ can either be a 1 or 0 with probabilities $p^1_{w_{l_1:l_3}}$ and $p^0_{w_{l_1:l_3}} = 1 - p^1_{w_{l_1:l_3}}$. Then $l_3$ can either be a 1 or 0 with probabilities $p^1_{w_{l_2:l_4}}$ and $p^0_{w_{l_2:l_4}} = 1 - p^1_{w_{l_2:l_4}}$. This continues for all letters in $L$, where $l_{n-1}$ is either a 0 or a 1 with probabilities $p^1_{w_{l_{n-2}:l_n}}$ and $p^0_{w_{l_{n-2}:l_n}} = 1 - p^1_{w_{l_{n-2}:l_n}}$. This produces the joint distribution, $\mathcal{L}(L|p) = p^{l_2}_{w_{l_1:l_3}} \times p^{l_3}_{w_{l_2:l_4}} \times ... \times p^{l_{n-1}}_{w_{l_{n-2}:l_n}}$, where by collecting like terms for each possible conditional word probability, the above result is given. Note that this is invariant to direction. The direction or labelling is arbitrary providing the letter ordering in the sequence remains. □

For any word length, $m \geq 1$, the likelihood in terms of the conditional word probabilities is given in Theorem 6.14. This again has the same form as that seen in Lemma 6.13 and for the directional version in Theorem 4.20.

**Theorem 6.14** (Conditional Word Likelihood, $m \geq 1$)**.**

$$\begin{aligned}\mathcal{L} &= \prod_{i=0}^{2^{2m}-1} \left(p^0_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}\right)^{n^0_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}} \\ &\qquad \times \left(p^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}\right)^{n^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}} \\ &= \prod_{i=0}^{2^{2m}-1} \left(1-p^1_{d(i)}\right)^{n^0_{d(i)}} \left(p^1_{d(i)}\right)^{n^1_{d(i)}}\end{aligned}$$

where, $d(i) = \frac{1}{2^m}\left[i - (i \bmod 2^m)\right] \,:\, i \bmod 2^m$

*Proof.* See Appendix B □

### 6.2.9 Inference

Given a sequence of 0's and 1's, we can estimate the word length and conditional word probabilities using Bayes' theorem and Bayes' factors (Kass and Raftery, 1995; O'Hagan, 1997) in a similar method to the directional de Bruijn version. As previously stated, it is possible to perform inference in a similar way to the directional de Bruijn process since the non-directional words can be converted to their directional de Bruijn word equivalents. The method of inference is also similar to that used by Besag (1986), but I choose to incorporate the neighbourhood property of de Bruijn graphs in the likelihood, rather than in the prior.

To estimate the conditional word probabilities, we can combine the likelihood from Theorem 6.14 with a beta prior, $P(p) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1}$ for $\alpha > 0, \beta > 0$, using Bayes' theorem. This is the same as was seen in Chapter 4, where we can incorporate any initial information we might have about the conditional word probabilities into the prior. I have again chosen a beta prior due to the conjugate relationship between the prior and likelihood. The posterior distribution for the conditional word likelihood for the non-directional de Bruijn process with word length $m = 1$ is as follows:

$$
\begin{aligned}
P(p|seq) &= \frac{\mathcal{L}(seq|p)P(p)}{\int \mathcal{L}(seq|p)P(p)dp} \\
&\propto \mathcal{L}(seq|p)P(p) \\
&= (1-p^1_{0:0})^{n^0_{0:0}} (p^1_{0:0})^{n^1_{0:0}} (1-p^1_{0:0})^{\beta_1-1} (p^1_{0:0})^{\alpha_1-1} \\
&\quad \times (1-p^1_{0:1})^{n^0_{0:1}} (p^1_{0:1})^{n^1_{0:1}} (1-p^1_{0:1})^{\beta_2-1} (p^1_{0:1})^{\alpha_2-1} \\
&\quad \times (1-p^1_{1:0})^{n^0_{1:0}} (p^1_{1:0})^{n^1_{1:0}} (1-p^1_{1:0})^{\beta_3-1} (p^1_{1:0})^{\alpha_3-1} \\
&\quad \times (1-p^1_{1:1})^{n^0_{1:1}} (p^1_{1:1})^{n^1_{1:1}} (1-p^1_{1:1})^{\beta_4-1} (p^1_{1:1})^{\alpha_4-1} \\
&= (1-p^1_{0:0})^{n^0_{0:0}+\beta_1-1} (p^1_{0:0})^{n^1_{0:0}+\alpha_1-1} \\
&\quad \times (1-p^1_{0:1})^{n^0_{0:1}+\beta_2-1} (p^1_{0:1})^{n^1_{0:1}+\alpha_2-1} \\
&\quad \times (1-p^1_{1:0})^{n^0_{1:0}+\beta_3-1} (p^1_{1:0})^{n^1_{1:0}+\alpha_3-1} \\
&\quad \times (1-p^1_{1:1})^{n^0_{1:1}+\beta_4-1} (p^1_{1:1})^{n^1_{1:1}+\alpha_4-1}
\end{aligned}
$$

Again, the posterior distribution for the de Bruijn process conditional word probabilities is a product of beta densities. Due to the conjugate relationship, the denominator in Bayes' theorem ($\int \mathcal{L}(seq|p)P(p)dp$) can easily be found using the identity, $\int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}$. This gives the model evidence as follows:

$$\begin{aligned}\int P(seq|p)P(p)dp =& \frac{\Gamma(n^0_{0:0}+\beta_1)\Gamma(n^1_{0:0}+\alpha_1)}{\Gamma(n^0_{0:0}+n^1_{0:0}+\beta_1+\alpha_1)} \times \frac{\Gamma(n^0_{0:1}+\beta_2)\Gamma(n^1_{0:1}+\alpha_2)}{\Gamma(n^0_{0:1}+n^1_{0:1}+\beta_2+\alpha_2)} \times \\ & \frac{\Gamma(n^0_{1:0}+\beta_3)\Gamma(n^1_{1:0}+\alpha_3)}{\Gamma(n^0_{1:0}+n^1_{1:0}+\beta_3+\alpha_3)} \times \frac{\Gamma(n^0_{1:1}+\beta_4)\Gamma(n^1_{1:1}+\alpha_4)}{\Gamma(n^0_{1:1}+n^1_{1:1}+\beta_4+\alpha_4)} \times\end{aligned}$$

The posterior distribution for the general case when $m \geq 1$ is given in Theorem 6.15.

**Theorem 6.15** (Posterior Distribution for de Bruijn Conditional Word Probabilities, $m \geq 1$)**.**

$$\begin{aligned}P(p|seq) &= \frac{\mathcal{L}(seq|p,m)P(p|m)}{P(seq)} \\ &= \frac{\mathcal{L}(seq|p,m)P(p|m)}{\int \mathcal{L}(seq|p,m)P(p|m)dp}\end{aligned}$$

where,

$$\begin{aligned}\mathcal{L}(seq|p,m)P(p|m) =& \prod_{i=0}^{2^{2m}-1} \left(1-p^1_{\frac{1}{2^m}[i-(i \bmod 2^m)]\,:\,i \bmod 2^m}\right)^{n^0_{\frac{1}{2^m}[i-(i \bmod 2^m)]\,:\,i \bmod 2^m}+\beta_{i+1}-1} \\ & \times \left(p^1_{\frac{1}{2^m}[i-(i \bmod 2^m)]\,:\,i \bmod 2^m}\right)^{n^1_{\frac{1}{2^m}[i-(i \bmod 2^m)]\,:\,i \bmod 2^m}+\alpha_{i+1}-1} \\ =& \prod_{i=0}^{2^{2m}-1} \left(1-p^1_{d(i)}\right)^{n^0_{d(i)}+\beta_{i+1}-1} \left(p^1_{d(i)}\right)^{n^1_{d(i)}+\alpha_{i+1}-1}\end{aligned}$$

and

$$\begin{aligned}&\int P(seq|p,m)P(p|m)dp \\ &= \prod_{i=0}^{2^{2m}-1} \frac{\Gamma\left(n^0_{\frac{1}{2^m}[i-(i \bmod 2^m)]\,:\,i \bmod 2^m}+\beta_{i+1}\right)\Gamma\left(n^1_{\frac{1}{2^m}[i-(i \bmod 2^m)]\,:\,i \bmod 2^m})+\alpha_{i+1}\right)}{\Gamma\left(n^0_{\frac{1}{2^m}[i-(i \bmod 2^m)]\,:\,i \bmod 2^m}+n^1_{\frac{1}{2^m}[i-(i \bmod 2^m)]\,:\,i \bmod 2^m}+\beta_{i+1}+\alpha_{i+1}\right)} \\ &= \prod_{i=0}^{2^{2m}-1} \frac{\Gamma\left(n^0_{d(i)}+\beta_{i+1}\right)\Gamma\left(n^1_{d(i)})+\alpha_{i+1}\right)}{\Gamma\left(n^0_{d(i)}+n^1_{d(i)}+\beta_{i+1}+\alpha_{i+1}\right)}\end{aligned}$$

with $d(i) = \frac{1}{2^m}\left[i - (i \bmod 2^m)\right] \ : \ i \bmod 2^m$

*Proof.* See Appendix B □

Due to the similarities between the directional and non-directional de Bruijn processes, the method to estimate the de Bruijn word length can again be treated in the same way for the non-directional process. Therefore, we still proceed to estimate the word lengths using Bayes' factors. Given a sequence of 0's and 1's, we will then be able to estimate both the conditional word probabilities and the word length of the non-directional de Bruijn process which is most likely to have generated it. The general method of Bayes' factors used is exactly the same as explained in Section 4.3, where we use the method to question whether a given sequence, $S$, was generated from one of two hypotheses. The first hypothesis says the sequence was generated from a de Bruijn process with word length $m_1$ and the second says the sequence was generated from a de Bruijn process with word length $m_2$. We have a probability for each of these occurring, along with prior probabilities that each of these sequences was indeed generated with the proposed word length. We can then produce the appropriate posterior probabilities, and if we consider Bayes' theorem in terms of an odds scale of these hypotheses in favour of $m_1$, we can define the Bayes' factor ratio as follows:

$$B_{1,2} = \frac{P(S|m_1)}{P(S|m_2)},$$

if the hypotheses, $m_1$ and $m_2$, are equally likely. As in Chapter 4, we can find an expression for $P(S|m_k)$ by integrating over the parameter space for the conditional word probabilities, such that:

$$P(S|m_k) = \int \mathcal{L}(S|p_k, m_k) P(p_k|m_k) \ dp_k,$$

for $k \in \{1, 2\}$, where $\mathcal{L}(S|p_k, m_k)$ is the likelihood of the data and $P(p_k|m_k)$ is the prior density of the model parameters, $p$. This gives the equation for the model evidence from Theorem 6.15, which is possible to calculate as we have a conjugate relationship between the likelihood and the prior. It is important to note here that the model evidence is independent of the conditional word probabilities, $p$. This

means that we are able to estimate the form of the de Bruijn process (the word length) before we make any attempt to estimate the $p$'s. To help with computation, I use the log scale. We calculate a Bayes' factor ratio, $B_{i,j} = \frac{P(S|m_i)}{P(S|m_j)}$, for each combination of models, where $i, j \in \{1, 2, ..., 10\}$, in order to find which model is preferable.

In the set up of the de Bruijn process, I make the assumption that the word length, $m$, cannot be too large. Otherwise, we are left with very large word lengths with far too many parameters to be estimated with little benefit to the model. Therefore, I make the decision to limit word lengths to not be any greater than 10. Since we have a conjugate prior, it is easy to calculate the evidence for each word length, compare each model in turn and determine the word length that best represents the given sequence. When values of $B_{i,j}$ are large, we have more evidence to reject the model with word length $m_i$ in favour of the model with word length $m_j$. The method of Bayes' factor for estimating the word length for non-directional de Bruijn processes is given in Theorem 6.16.

**Theorem 6.16** (Estimation of Non-Directional De Bruijn Word length by Bayes' factors, $m \geq 1$)**.** Consider a sequence of 0's and 1's which was generated under one of two hypotheses. The first is a de Bruijn process with word length $m_1$ and the second is a de Bruijn process with word length $m_2$. The Bayes' factor ratio is as follows:

$$B_{1,2} = \frac{P(S|m_1)}{P(S|m_2)}$$

where,

$$\begin{aligned} P(S|m_k) &= \int P(seq|p, m_k) P(p|m_k) dp \\ &= \prod_{i=0}^{2^{2m}-1} \frac{\Gamma\left(n^0_{d(i)} + \beta_{i+1}\right) \Gamma\left(n^1_{d(i)}) + \alpha_{i+1}\right)}{\Gamma\left(n^0_{d(i)} + n^1_{d(i)} + \beta_{i+1} + \alpha_{i+1}\right)}, \end{aligned}$$

for $k \in \{1, 2\}$ and $d(i) = \frac{1}{2^m} \left[i - (i \bmod 2^m)\right] \ : \ i \bmod 2^m$

When values of $B_{1,2}$ are large, we have more evidence to reject the first hypothesis with word length $m_1$ in favour of the second hypothesis with word length $m_2$.

*Proof.* Follows from Theorem 6.15. □

Once we have estimated the word length, we are then able to estimate the conditional word probabilities by applying the result in Theorem 6.15.

## 6.3 2-d Methodology

Following on from the 1-d methodology for non-directional de Bruijn processes, it is now important to consider what form 2-d non-directional de Bruijn processes might take. The main inspiration for non-directional de Bruijn processes came from considering 2-d directional de Bruijn processes, where applying a directionality did not strictly make sense. There has been a large amount of previous research done in this area, particularly within the scope of Markov random fields and image processing, including Besag (1986). In this current section, I have outlined how I would proceed in solving the problem based on the non-directional de Bruijn process established in this chapter. For future work, I would like to look at comparisons with existing methods, and determine whether we can use any existing methodology or properties to improve the ideas further.

One of the main objectives is to find a form of the word so that, like the 2-d directional de Bruijn process (method 2) and the 1-d non-directional de Bruijn graph, we can convert the words into their 1-d directional word form. This is providing that we can decompose each of the 2-d words in the same way to give a 1-d vector. We can then use the methodology from Section 4.3 for inference to estimate both the conditional word probabilities and the word length.

When considering what the form of the word could take for the 2-d non-directional case, there are many different options. Not only do we want to define the shape or form of the word, but we also want to classify different sizes of the word so that we can have different 'word' length de Bruijn processes. When considering different sizes of words for different types of 2-d de Bruijn words, we can start to form families of words. I define a family of words as all of the possible words for a de Bruijn process for each word length $m$. For example, the words defined in Figure 5.2.2 for Method 2 2-d directional de Bruijn process would define a family. I chose this form of the word to best include all of the letters that the current point may be dependent on. Equally,

I could have changed the form of the word to include all letters in an L shape behind the current letter. Where, for each $m$, the next L row of letters would be added to the current letters to form the next word. All of these words for each word of length $m$ would form another family of words for the 2-d directional de Bruijn process. Any family of words can be chosen providing the words in each family are nested (the smaller words must be contained in the larger sized words).

Two different possible families of 2-d non-directional de Bruijn words are shown in Figure 6.4. Here, there are two fairly similar families of words that are formed from diamond or square patterns in the grids of letters. The grey points represent either 0's or 1's, and the green point is the current point we are interested in estimating. Initially, when considering what the first word might look like for either of these cases, the simplest option is to take the four closest points in a cross shape. These are outlined by the red lines in both plots in Figure 6.4, representing size $m = 1$ words in both cases. Now, considering what points might make up the size $m = 2$ word, we could include the additional four corner points to produce a square. This is outline by the purple line in the right hand plot. Alternatively, as shown in the left hand plot, the size $m = 2$ word (purple) could be chosen to follow the same diamond pattern as the $m = 1$ word with an increased diameter. The remaining words in this family would follow this pattern by increasing the size of the diamond with increasing word size. Each word is formed by including letters which are $m$ points away from the green point moving horizontally or vertically (city block distance).

A third possible family of words is shown in Figure 6.5. This family was created using the Euclidean distance taken from the green point to the letters in the neighbourhood. The word of size $m = 1$ then consists of the four closest letters to the green point in a cross. Larger words are then obtained by adding in the next closest points with respect to the Euclidean distance.

Given that we have generated a family of 2-d words for the de Bruijn process, it would then be ideal if we had a formula for generating the coordinates of the points in the next word size given the current word. My current idea is to generate a recurrence relationship, or generating rule, so that we would have a general formula

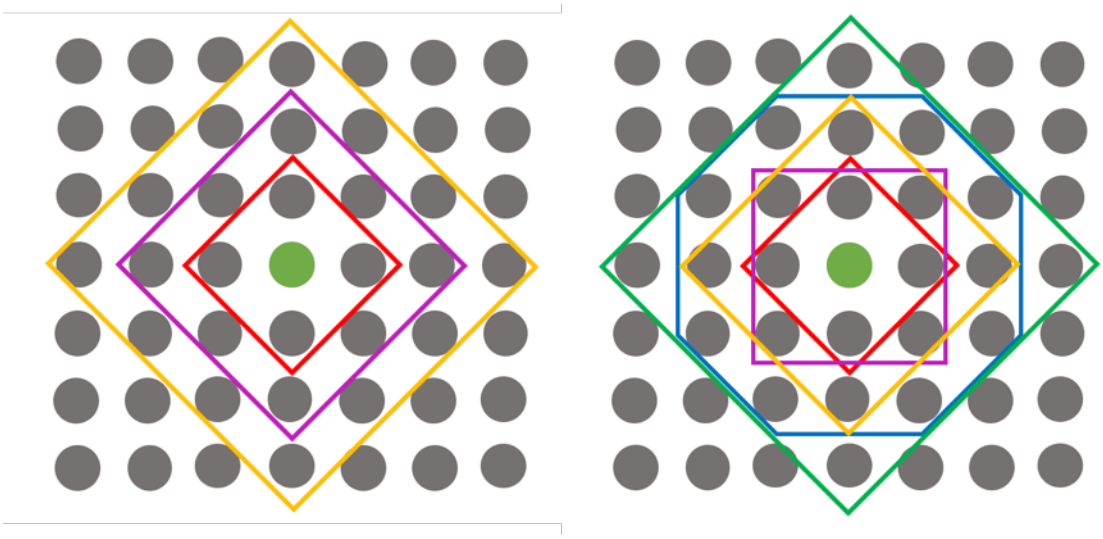


Fig. 6.4 Two examples of 2-d non-directional de Bruijn word families, where the green point is the point of interest to be simulated. Grey points represent either 0's or 1's. The forms of the words for word sizes $m = 1$, $m = 2$, $m = 3$, $m = 4$ and $m = 5$ that the green point is dependent on are outlined in red, pink, yellow, blue and green respectively. The words in the left plot are formed from letters which are $m$ points away from the green point moving vertically and horizontally.

for producing the individual words for each family. For example, if the green points in Figure 6.4 have coordinates $(a, b)$, then we can calculate formulas to generate the coordinates for all of the letters included in a size $m$ de Bruijn word. For the diamond word family on the left in Figure 6.4, the coordinates, $(x, y)_m$, for the included letters in a size $m$ word are given by,

$$(x, y)_m = (a, b) + C_m$$

where,

$$C_m = \{(m, 0), (-m, 0)\} \cup \{(0, \pm m) + (bn, \mp n) : b \in \{-1, 1\}, 0 \geq n \geq m - 1\}.$$

For the word family in Figure 6.5 where the words are generated using Euclidean distance, the coordinates, $(x, y)_m$, for the letters in a size $m$ word are given by,

$$(x, y)_m = C_{m-1} + C_m$$

where,

$$C_m = \{(a, b) : \sqrt{a^2 + b^2} \leq \sqrt{x^2 + y^2}$$
$$\forall\ (x, y), (a, b) \in (X, Y),\ \text{ where } (x, y) \notin C_{m-1}\}.$$

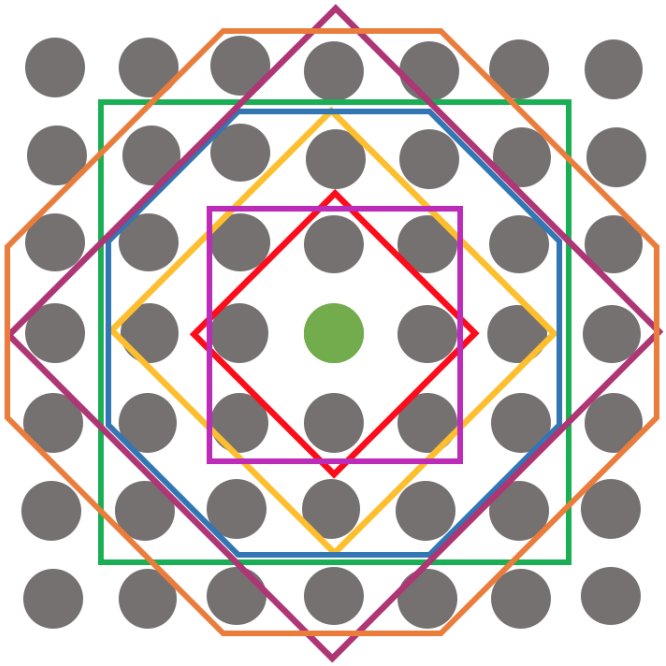


Fig. 6.5 Example of a 2-d non-directional de Bruijn word family generated using Euclidean distance, where the green point is the point of interest to be simulated. Grey points are either 0's or 1's. The forms of the words for word sizes $m = 1$, $m = 2$, $m = 3$, $m = 4$, $m = 5$, $m = 6$ and $m = 7$ that the green point is dependent on are outlined in red, pink, yellow, blue, green, purple and orange respectively.

where $(X, Y)$ is the set of all possible coordinates in the predefined grid.

Unfortunately, I still have the same difficulties in simulating sequences from 2-d non-directional de Bruijn processes as I did for the 1-d version. This is thus a high priority future problem so that we can easily view samples from 1-d and 2-d de Bruijn processes. There is also a combinatoral problem with the word sizes for all of the families proposed above. As the word size increases, the number of letters in each word quickly gets very large, requiring enormous numbers of potential words and conditional word probabilities. Hence, for future work it would be a good idea to try and reduce this number somehow by introducing further constraints. For example, we may be able to define the conditional word probability to be dependent on how many 1's or 0's there are in each word. As well as this, we should also be able to reduce the number of parameters by taking advantage of symmetries in the word and marginal probabilities.

A useful tool for future work with non-directional de Bruijn processes may be graph theory. Using graph theory, we may be able to write the de Bruijn words

themselves as graphs and apply well known identities. This may then help us tackle the simulation problem, or give interesting properties such as a 2-d equivalence to run length. In graph theory, there is a Markov property known as conditional independence (Lauritzen, 1996). This property states that a set of nodes in a graph can be conditionally independent of another set of nodes given dependence of a third set of nodes. If we apply this to our de Bruijn words in Figure 6.4, then we could say that the green point which we are interested in, conditional on the outer letters in the word, is independent of all other letters. In other words, the centre letters are independent of each other conditional on knowing the outer letters of the word. Hopefully this would then give the joint distribution over all of the letters in the word, where some are independent of each other. Then, given this joint distribution, we may be able to decompose it into a set of conditionally independent objects. A lot of the theory behind this is described in the literature as 'total positivity', where we would be particularly interested in total positivity in binary distributions (Lauritzen et al., 2019).

Hence, I believe a deeper understanding of graph theory will be a vital part of future work on de Bruijn processes to see if there are any important properties that we may be able to take advantage of. This may be useful when considering patch size, or the 2-d equivalent to run length. For example, we may have the diamond de Bruijn word family shown in the left plot in Figure 6.4, and look to define patch run sizes also in a diamond shape. The runs we are trying to observe are diamond shapes consisting entirely of 1's with a ring of 0's around the edge. Therefore, using conditional independence, conditional on the 0's surrounding a diamond of 1's, the 1's are independent of everything else. Conditional on the boundary of 0's, we should then be able to calculate the probability of every letter within the diamond being 1. This provides the conditional probability, where we would then need to provide the probability of getting the ring of 0's for the full joint distribution.

## 6.4 Discussion

In this chapter, I have given an overview of one possible method for a non-directional de Bruijn process. The inspiration for this work came from looking at 2-d directional de Bruijn processes, where I argued that the direction used in de Bruijn graphs did not make logical sense in a 2-d input space. Inspiration was taken from existing literature including Abend et al. (1965); Agapie et al. (2014, 2004); Banerjee et al. (2004); Besag (1974, 1986); Wolfram (2002).

The non-directional de Bruijn process presented here is based on a 1-d directional de Bruijn process, so that many of the properties and method of inference could be carried across. Therefore, I chose the form of the de Bruijn word so that they could be written in terms of their 1-d directional de Bruijn word equivalents. I was then able to define a run length distribution and give definitions for the expectation, variance and generating functions for the run length distribution. The method of inference is also outlined where we are again able to use the method of Bayes' factors with either maximum likelihood or Bayesian inference to estimate both the word length and associated conditional word probabilities.

One of the major issues with the current non-directional methodology is simulation. I currently do not have a successful method of simulating a sequence from a given de Bruijn method. I have outlined two possible solutions for this, but they both either have errors in the formulation or have high combinatorial problems. This is one area of this thesis that I am keen to rectify in future work.

Another area for future work is the run length distribution. Although I have fully defined the run length distribution for the $m = 1$ length word, the run length distribution for larger word lengths is not complete. Due to the run-in period at both the start and end of each run of 1's (or 0's), we must average over all possible combinations of start and end sequences to give the full distribution. This results in requiring the joint marginal probabilities for each of these sequence combinations in terms of the length $m$ de Bruijn word. I have not currently found a solution for this, but have considered whether it would be possible to form a recurrence relationship to

define the marginal probability of a smaller word in terms of the marginal probability of a larger word.

In Section 6.3 I also gave a brief introduction on how we might be able to solve the 2-d non-directional de Bruijn process problem. This was mainly based on trying to define the form of the word for such a process, which we can then decompose into the 1-d directional de Bruijn process so that we can use the methodology developed in Sections 3 and 4. To do this, I have introduced the concept of a family of words which defines each size of the de Bruijn word in the process. This is a challenging topic, so the majority of this is left for future work.

Overall, this method of non-directional de Bruijn processes is not necessarily the best way to view this problem, but it is my initial thoughts, designed such that it links in well to the work already accomplished in the thesis. Therefore, a main aim of future work is to look back at this concept of non-directionality to improve on the method outlined here and potentially seek alternative methods.

# Chapter 7

# Conclusions and Future Work

The main motivation for this thesis was uncertainty quantification for numerical models where there are two or more output solutions which are separated by distinct boundaries. Given that these types of models can produce discontinuities between the distinct regions, it was important not to assume any continuity across the entire input space, as has been done in previous work such as treed Gaussian processes Gramacy and Lee (2009). This constraint meant that it was infeasible to fit a single Gaussian process emulator to the whole of the model. It was also decided that the model outputs could take any form, being either qualitative or quantitative. Hence, I moved towards modelling the separate regions using a classification method.

The main problem with current classification methods, such as logistic regression, is the lack of any distance correlation when drawing from an independent Bernoulli distribution to give classification predictions. Therefore, in Chapter 2 of this thesis, I detailed one method for producing a correlated classifier using a latent Gaussian process. Then, in Chapters 3 to 6, I outlined an alternative methodology for a correlated Bernoulli process using de Bruijn graphs. The rest of this chapter gives an overview of each previous chapter along with suggestions for future work.

Chapter 2 outlined a new method for classifying models or simulators with two distinct regions. The proposed method is an improvement on current methods of classifications, such as logistic regression, as it includes correlation through a distance metric and can be applied to a wide range of applications. The model is based on a class labelling system where initial points are labelled according to

which known region they lie in. We were then able to model the labelling function instead of the function itself. This is important because we are now able to deal with a much broader range of applications where outputs to the model are not necessarily quantitative. The labelling function becomes latent in the model and can be modelled as a Gaussian process, hence introducing the necessary correlation into the model. This latent Gaussian process was estimated using Metropolis Hastings MCMC. The prior specification is very important to the model, and model validation was included using a misclassification rate based on a leave-one-out cross-validation. Several examples are given, including a motivating example based on analysing the reproductive system in mammals. This bimodel system has two regions in output space corresponding to high and low rates of reproduction.

Following the success of the motivating example, I am keen to test the method out on more real life data sets to see how well it performs for more intricate output regions. The example in Figure 2.17 had a fairly simple linear boundary between regions, and so I would like to test out the limits of the method for more complex region shapes. I have already shown that my method performs well with a doughnut shape, so I have confidence that it will be able to cope in other scenarios. As well as this, I would like to work with examples in higher dimensions. I have tested the method out on toy examples in 1d, 2d and 3d, but it would be interesting to see if the classifier would struggle if we increased the dimensions further. As the dimensions increase, we would need more initial points, and a clear idea of how many different regions we expect to get a sufficient boundary estimate. I expect prior specifications will become increasingly important as we move up in dimensions.

Following on from this, I would also like to experiment with examples that have more than two distinct output regions. I have already briefly outlined how I would initially tackle this problem, but I would like to take this further to produce an actual methodology. If possible, I would like to adapt the latent Gaussian process so that it could cope with more output regions, but I expect this to be a difficult problem due to the negative/positive condition that is vital for the classification.

In Section 2.6 I discussed the comparisons between the latent Gaussian process method, and logistic regression and Voronoi tessellation (naive approach). In both of these cases, it is clear that my method outperforms the alternative. In the future, I would like to make comparisons with other methods that are currently being developed, where it is not necessarily guaranteed that my new method will outperform. An example of this is Pope et al. (2018), who model spatial processes with heterogeneity or discontinuities by using a combination of Voronoi tessellation and Gaussian processes. If I were to also compare my method to alternative classification methods in machine learning literature, I may be able to use their tools to adapt and improve my method further. I also hope to improve the efficiency of my computer code. For example, I could look into changing the MCMC to Hamiltonian MCMC to speed up calculations.

Lastly, there is still a significant contribution that could be made to the design problem. In Section 2.7, I briefly discussed a possible way to solve this problem, but it may not be the most efficient. A design method is required to improve the accuracy of the classification and the boundary estimation with limited initial data. Since the Gaussian process is applied to the labelling function, $\Lambda$, instead of the actual function output itself, we find that most current design methods are not applicable. Therefore, I would like to either continue developing my current design method, or seek out an entirely new method that may be more efficient in improving the classification. It would also be ideal if we could create an optimiser (similar to the improvement function used by Ranjan et al. (2008)) that would indicate whether we should be choosing points to be more space filling or to improve the estimate for the boundary. Currently, I am only choosing one new point at a time in the design, but I would like to expand this to be able to select multiple new points at a time.

Chapters 3 and 4 introduced a correlated Bernoulli process using de Bruijn graphs. The aim of this work was to produce an equivalence to a Bernoulli process where correlation is incorporated when making draws or samples: if we had a sample sequence of 0's and 1's, then we would like to see like symbols cluster together instead of appearing at random. Eventually, I would like to use this method alongside a

classification method such as logistic regression so that, when we draw 0's and 1's to make classification predictions, correlation with respect to distance is accounted for. I am keen to see if I can use the methodology developed in these two chapters for a real life data set with classification.

The method that I have proposed is based on de Bruijn graphs, which are directed graphs that have a Markov property on the de Bruijn 'word' but not on the 'letter'. A de Bruijn word of length $m$ consists of any combination of 0's and 1's of length $m$. The length of the de Bruijn word allows us to control the spread of the correlation between neighbouring letters in a sequence; i.e. the length scale. Using the defined de Bruijn process we are able to generate sequences that appear very 'sticky', as well as 'anti-sticky' sequences that are much more structured, and alternate between 0's and 1's.

In Chapter 4, I looked into the properties of de Bruijn processes, including the run length distribution. The run length distribution gives the probability of a run of 1's of length $n$ bounded by a 0 at each end. From this, we were then able to calculate the expected run length, variance of run length and generating functions. Lastly, I discussed the inference for de Bruijn graphs so that, given a sequence of 0's and 1's, we can estimate both the word length $m$ and the associated transition probabilities. This was achieved using Bayes' factors accompanied with either maximum likelihood or Bayesian inference.

There are many areas for future work on the topic of de Bruijn graphs and correlated Bernoulli processes. To start with, there is a lot of progress that could be made by looking at group theory. Since we are incorporating well known structures like Markov chains, there are many theorems and identities that we could apply to the de Bruijn process. Such theorems would hopefully enable me to improve on the definitions by making simplifications and help in expanding de Bruijn graphs to higher dimensions. Hence, I think that it would be advantageous to dedicate some research into linking de Bruijn graphs to group theory, graph theory and extensive Markov chain theory. As well as this, there are also many properties that come from the eigenvalues and eigenvectors of Markov chains. Although I have considered them

briefly, there is still plenty of extra work that could be done, including whether we could find any more general definitions, such as the stationary distribution and chain conversion rates. This also includes problems similar to a k-out-of-n problem or an extreme event problem. For example, the sequence 111 may be an extreme event and we would like to know the probability of it occurring in a given sequence.

Secondly, it has stood out that de Bruijn graphs can quickly become very complicated with many possible transition probabilities for large word lengths. Therefore, I believe that it would be useful to try and limit the number of transition probabilities, which could be particularly important in inference (Chapter 4). I have considered reducing the probabilities by making the probability of getting a 1 next in the sequence be dependent on how many 1's there are currently in the word before. In other words, we would have fewer transition probabilities that were functions of the proportion of 1's in a small window or word. Even if this simplification were successful, we would have to be careful not to reduce the flexibility too much. If we set it so that the probability of getting a 1 was solely dependent on how many 1's there were in the previous word, then the word 011 would have the same probability of getting a 1 as the word 101. Therefore, we would ideally like to reduce the number of parameters whilst also maintaining the flexibility that I have incorporated so far.

Although mentioned briefly, I also intend to develop non-stationary de Bruijn processes further. The transition probabilities associated with the de Bruijn process will depend on the time step of the Markov chain. Hence, we would be able to create chains that are sticky in some places and not sticky in others. This will be particularly important when trying to apply the de Bruijn process to an actual classification problem. The process will act sticky for areas away from the region boundary, where we are sure of the classification. However, the process will then be similar to Bernoulli trials close to the boundary where we have little information for where the exact boundary lies. We should then hopefully be able to produce something similar to a step function, with 0's corresponding to one region and 1's corresponding to the other. There will probably still be a small fuzzy section near the border, but the results will be an improvement on logistic regression (such as that in Figure 1.3).

One of the main challenges for non-directional de Bruijn processes is that we can no longer use many of the Markov and geometric distribution properties that were used to help simplify the expressions describing stationary de Bruijn processes. This includes difficulties in finding stationary distributions, and calculating subsequent measures for the run length distribution.

One of the areas I would also like to research is the accuracy of the word length and transition probability estimates. Currently in my examples, I am able to assess the accuracy of the estimates by comparing them against the true values that I simulated the sequences with. This is not going to be possible for real life examples and so I would like to establish a method of validation to give a level of uncertainty in the estimates. A more extensive research into Markov properties may be able to help me solve this problem.

Chapter 5 expanded the ideas from Chapters 3 and 4 in an attempt to produce a 2-d de Bruijn process. This was a very challenging problem since the vital direction in de Bruijn graphs does not make logical sense on a two-dimensional grid. I outlined two possible methods for producing a 2-d de Bruijn process. Method 1 was based on multivariate Markov chains which allowed the rows in a 2-d grid to have a notable correlation between them. The rows are connected using the de Bruijn word length so that each point is dependent on de Bruijn graphs in three directions. Although simulation was possible, the method required high numbers of parameters to be estimated and inference would be challenging.

Method 2 proved to be more successful, and enabled us to perform inference. The aim of Method 2 was to be closer aligned to the 1-d de Bruijn process already developed, with the hope that some of the methodology could be applied. This was possible by changing the form and shape of the de Bruijn word. Although it is no longer possible to visualise the associated de Bruijn graph, it is possible to use the inference methodology developed in Chapter 4. By converting each 2-d de Bruijn word into a vector, we can find the 1-d de Bruijn word equivalent. We can then use the method of Bayes' factors to estimate the word length, along with either maximum likelihood or Bayesian inference to find the associated transition probabilities.

One area of improvement for this method would be to reduce the large numbers of transition probabilities that occur with increasing word size. This was mentioned briefly in Chapter 4, but it would be ideal if we could reduce the number of parameters further. Expanding Method 2 to higher dimensions was also discussed, where I proposed how I would tackle a three dimensional de Bruijn process. As future work, I would like to attempt to simulate a 3-d example and also attempt to adapt the method further to higher dimensions. I believe this to be possible providing we can state the 1-d de Bruijn word equivalent to an $n$-d word.

I am particularly interested in developing the 2-d equivalent to the run length distribution introduced in Chapter 4. This will involve defining what a patch size is on a 2-d grid, since it is possible to have both circular patches and long chains or 1's (or 0's). If we just define a patch size by counting how many 1's (or 0's) that are touching at least one other 1 (or 0), then these patches would technically be equivalent. The main focus of this work will be to develop a formal definition of a patch size. As well as this, we could also consider the structures of diagonal lines of 1's (or 0's) across a 2-d grid. I am also considering whether 2-d de Bruijn processes have to stay on a rectilinear grid. If we had the 0's and 1's laid out in a hexagonal grid, then it is clear that each point has six nearest neighbours. This may then help in defining what a patch size is.

It would also be ideal to extend de Bruijn processes to infinite dimensions where currently, I have only focused on methodology in finite dimension. If we had a grid of points, and the points then get so close together that we have something more like a correlation function rather than a matrix. However, this is a very hard problem to solve. It may be the case that the infinite and finite cases have a similar relationship to a Gaussian Markov random field and a Gaussian process. This conversion acts to transform something that is split into blocks, into something that is then continuous. Therefore, I believe that further research into non-Gaussian Markov random fields will be of great importance for future work.

One of the major disadvantages of the proposed 2-d de Bruijn process is the directionality that was imposed. The main aim of Chapter 6 was to remove this

direction to produce a non-directional de Bruijn process. I started by developing the 1-d methodology, where I again chose to base the 1-d non-directional de Bruijn process on the already developed 1-d directional de Bruijn process. This meant changing the form of the word so that each non-directional word could be written in terms of their 1-d directional de Bruijn word equivalent. From this, it was then possible to define a run length distribution, expected run length, variance of run length, generating functions and inference through Bayes' factors. The topic of non-directional de Bruijn processes is of great interest to me, and I am eager to pursue the research further following on from this thesis.

Currently, the de Bruijn processes described in this chapter have no successful method of simulating a sequence of 0's and 1's. Two possible methods are outlined, but they either contain an error or are very computationally inefficient. This would be one of my initial areas to concentrate on in future work, as it is important that we can view sample sequences from certain de Bruijn processes. Given a simulated sequence, we will also be able to test the inference stated in the chapter. I have not attempted to apply the inference method yet due to being unable to generate a specific sequence with the correct correlations between letters.

Another important area for future work is the run length distribution. The run length distribution is successfully given for length $m = 1$ de Bruijn processes, but is incomplete for larger word lengths. I have not yet defined the marginal probabilities for the burn-in sequences of 0's and 1's that occur at both ends of a run of 1's (or 0's). The major problem with this is defining the probability of a short sequence, given that we are working with a de Bruijn process with word length $m$. Once I have defined this (as well as improving the method of simulation), we will be able to use the run length distribution to compare both analytical and simulated run lengths from sequences generated from both sticky and non-sticky de Bruijn processes. Currently, I plan to calculate these marginal probabilities using a recurrence relationship to define the marginal probability of a smaller word in terms of the marginal probability of a larger word.

The overall aim for this work is to produce a non-directional de Bruijn process for any finite dimension. I have given a brief introduction on how I would proceed in trying to develop a 2-d non-dimensional de Bruijn process, but a lot of further research is required to develop the methodology. I have introduced the concept of a family of words which defines each size of the de Bruijn word in the process. Using this and an improved knowledge of graph theory, I believe the described goals can be achieved.

Overall, I have only skimmed the surface of non-directional de Bruijn processes and believe there is an extensive field of potential research. For example, the ideas presented in Chapter 6 may not necessarily be the best way to view this problem, and we may need to start from scratch to progress further. I have also generated ideas on possibly linking this work to that by Teugels (1990) to produce a more general Bernoulli process. As with the previous chapters, I am keen to progress further with de Bruijn processes and will hopefully one day apply the concepts to some real world examples.

# References


Abend, K., Harley, T., and Kanal, L. (1965). Classification of binary Random Patterns. *IEEE Transactions on Information Theory*, 11(4).

Agapie, A., Andreica, A., and Giuclea, M. (2014). Probabilistic Cellular Automata. *Journal of Computational Biology*, 21(9).

Agapie, A., Hons, R., and Muhlenbein, H. (2004). Markov Chain Analysis for One-Dimensional Asynchronous Cellular Automata. *Methodology and Computing in Applied Probability*, 6.

Andrianakis, I., Vernon, I. R., Mccreesh, N., Mckinley, T. J., Oakley, J. E., Nsubuga, R. N., Goldstein, M., and White, R. G. (2015). Bayesian History Matching of Complex Infectious Disease Models Using Emulation : A Tutorial and a Case Study on HIV in Uganda. *PLoS Computational Biology*, 11(1).

Ayyer, A., Bouttier, J., Corteel, S., and Nunzi, F. (2015). Multivariate Juggling Probabilities. *Electronic Journal of Probability*, 20(5).

Ayyer, A. and Strehl, V. (2011). Stationary Distribution and Eigenvalues for a De Bruijn Process. *ArXiv*.

Ba, S. and Joseph, R. (2012). Composite Gaussian Process Models for Emulating Expensive Functions. *The Annals of Applied Statistics*, 6(4):1838–1860.

Banerjee, S., Carlin, B. P., and Gelfand, A. E. (2004). *Hierarchical Modeling and Analysis for Spatial Data*. Chapman & Hall/CRC, United States of America, first edition.

Bastos, L. S. and O'Hagan, A. (2009). Diagnostics for Gaussian Process Emulators. *Technometrics*, 51(4):425–438.

Bect, J., Ginsbourger, D., Li, L., Picheny, V., and Vazquez, E. (2012). Sequential Design of Computer Experiments for the Estimation of a Probability of Failure. *Stat Comput*, 22:773–793.

Besag, J. (1974). Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):192–236.

Besag, J. (1986). On the Statistical Analysis of Dirty Pictures. *Journal of the Royal Statistical Society Series B*, 48(3):259–302.

Billingsley, P. (1961). Statistical Methods in Markov Chains. *The annals of Mathematical Statistics*, 32(1).

Brooks, S., Gelman, A., Jones, G., and Meng Xiao-Lin (2012). *Handbook of Markov Chain Monte Carlo*. Chapman & Hall/CRC, first edition.

Caiado, C. C. S. and Goldstein, M. (2015). Bayesian Uncertainty Analysis for Complex Physical Systems Modelled by Computer Simulators with Applications to Tipping Points. *Communications In Nonlinear Science and Numerical Simulation*, 26:123–136.

Chan, A. B. and Dong, D. (2011). Multivariate Generalized Gaussian Process Models. *IEEE Conference on Computer Vision and Pattern Recognition*.

Chang, W., Applegate, P. J., Haran, M., and Keller, K. (2014). Probabilistic Calibration of a Greenland Ice Sheet Model using Spatially Resolved Synthetic Observations: Toward Projections of Ice Mass Loss with Uncertainties. *Geoscientific Model Development*, 7:1933–1943.

Chang, W., Haran, M., Applegate, P., and Pollard, D. (2016). Calibrating an Ice Sheet Model using High-dimensional Binary Spatial Data. *Journal of the American Statistical Association*, 111(513).

Chib, S. and Greenberg, E. (1995). Understanding the Metropolis-Hastings Algorithm. *The American Statistician*, 49(4):327–335.

Ching, W.-k. and Ng, M. K. (2006). *Markov Chains: Models, Algorithms and Applications*. Springer Science + Business Media, Inc., New York, USA, first edition.

Ching, W.-k., Ng, M. K., and Fung, E. S. (2008). Higher-order Multivariate Markov Chains and their Applications. *Linear Algebra and its Applications*, 428:492–507.

Compeau, P. E. C., Pevzner, P. A., and Tesler, G. (2017). Why are De Bruijn Graphs Useful for Genome Assembly? *Nature biotechnology*, 29(11):987–991.

Craig, P., Goldstein, M., Seheult, A., and Smith, J. (1996). Bayes Linear Strategies for Matching Hydrocarbon Reservoir History. *Bayesian Statistics*, 5:69;95.

Craig, P. S., Goldstein, M., Rougier, J. C., and Seheult, A. H. (2001). Bayesian Forecasting for Complex Systems Using Computer Simulators. *Journal of the American Statistical Association*, 96(454):717–729.

Currin, C., Mitchell, T., Morris, M., and Ylvisaker, D. (1991). Bayesian Prediction of Deterministic Functions, With Applications to the Design and Analysis of Computer Experiments. *Journal of the American Statistical Association*, 86(416):953–963.

Dai, B., Ding, S., and Wahba, G. (2013). Multivariate Bernoulli Distribution. *Bernoulli*, 19(4):1465–1483.

De Bruijn, N. G. (1946). A Combinatorial Problem. *Communicated by Prof. W Van Der Woude*.

Dempster, A., Laird, N., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society Series B (Methodological)*, 39(1):1–38.

Diggle, P. J., Tawn, J. A., and Moyeed, R. A. (1998). Model-Based Geostatistics. *Appl. Statist.*, 47(3):299–350.

Do, C. B. and Batzoglou, S. (2008). What is the Expectation Maximization Algorithm? *Nature biotechnology*, 26(8).

Edwards, N. R., Cameron, D., and Rougier, J. (2011). Precalibrating an Intermediate Complexity Climate Model. *Climate Dynamics*, 37:1469–1482.

Feller, W. (1950). *An Introduction to Probability Theory and Its Applications*. John Wiley and Sons Inc., United States of America, third edition.

Fredricksen, H. (1992). A New Look at the De Bruijn Graph. *Discrete Applied Mathematics*, 37(38):193–203.

Fu, A. J. C. and Koutras, M. V. (1994). Distribution Theory of Runs : A Markov Chain Approach. *Journal of the American Statistical Association*, 89(427):1050–1058.

Gallier, J. (2008). Notes on Convex Sets, Polytopes, Polyhedra Combinatorial Topology, Voronoi Diagrams and Delaunay Triangulations. *ArXiv*.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). *Bayesian Data Analysis*. Chapman & Hall/CRC Press, Boca Raton, Florida, US, third edition.

Gilks, W. R., Richardson, S., and Spiegelhater, D. J. (1996). *Markov Chain Monte Carlo in Practice*. Chapman & Hall, London, 1 edition.

Golomb, S. W. (1967). *Shift Register Sequences*. World Scientific Publishing Co., London, third edition.

Good, I. J. (1946). Normal Recurring Decimals. *Journal of the London Mathematical Society*, 21(3).

Gosling, J. P. (2005). Elicitation : A Nonparametric View. *PhD Thesis, University of Sheffield*.

Gosling, J. P., Oakley, J. E., and O'Hagan, A. (2007). Nonparametric Elicitation for Heavy-Tailed Prior Distributions. *Bayesian Analysis*, 2(4):693–718.

Gramacy, R. B. and Lee, H. K. H. (2009). Bayesian Treed Gaussian Process Models with an Application to Computer Modeling. *Journal of the American Statistical Association*, 103(483):1119–1130.

Green, P. J. (1995). Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination. *Biometrika*, 82:711–732.

Harrell, F. E. (1979). Statistical Inference for Censored Multivariate Distributions Based on Induced Order Statistics. *PhD Thesis, University of North Carolina*.

Harrell, F. E. and Sen, P. K. (1979). Statistical Inference for Censored Bivariate Normal Distributions Based on Induced Order Statistics. *Biometrika*, 66(2):293–298.

Hauge, E. R. and Mykkeltveit, J. (1996). On the Classification of DeBruijn Sequences. *Discrete Mathematics*, 148:65–83.

Haylock, R. G. E. and O'Hagan, A. (1996). On Inference for Outputs of Computationally Expensive Algorithms with Uncertainty on the Inputs. *Bayesian Statistics*, 5.

Hilbe, J. M. (2009). *Logistic Regression Models*. Chapman & Hall/CRC, Boca Raton, Florida, US, first edition.

Hourdin, F., Mauritsen, T., Gettelman, A., Golaz, J. C., Balaji, V., Duan, Q., Folini, D., Ji, D., Klocke, D., Qian, Y., Rauser, F., Rio, C., Tomassini, L., Watanabe, M., and Williamson, D. (2017). The Art and Science of Climate Model Tuning. *American Meteorological Society*, 98(3):589–602.

Isaacson, D. L. and Madsen, R. W. (1976). *Markov Chains: Theory and Applications*. John Wiley and Sons Inc., New York, USA, first edition.

Johnson, N. L., Kemp, A. W., and Kotz, S. (2005). *Univariate Discrete Distributions*. John Wiley and Sons Inc., Hoboken, New Jersey, third edition.

Jones, P. W. and Smith, P. (2001). *Stochastic Processes: An Introduction*. CRC Press, Tayloe & Francis Group, Boca Raton, Florida, US, third edit edition.

Kass, R. E. and Raftery, A. E. (1995). Bayes Factors. *Journal of the American Statistical Association*, 90(430):773–795.

Kennedy, M. C. and O'Hagan, A. (2001). Bayesian Calibration of Computer Models. *Journal of the Royal Statistical Society. Series B (Methodology)*, 63(3):425–464.

Kim, H.-M., Mallick, B. K., and Holmes, C. C. (2005). Analyzing Nonstationary Spatial Data Using Piecewise Gaussian Processes. *Journal of the American Statistical Association*, 100(470):653–668.

Kleinbaum, D. G. and Klein, M. (1994). *Logistic Regression*. Springer, New York, NY, New York, 3 edition.

Knudde, N., Couckuyt, I., Dhaene, T., and Shintani, K. (2019). Active Learning for Feasible Region Discovery. *18th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 567–572.

Kotz, S., Balakrishnan, N., and Johnson, N. L. (2000). *Continuous Multivariate Distributions*. John Wiley and Sons, Inc., New York, USA, second edition.

Lauritzen, S., Uhler, C., and Zwiernik, P. (2019). Total positivity in structured binary distributions. *ArXiv*.

Lauritzen, S. L. (1996). *Graphical Models*. Oxford University Press, New York, USA, first edition.

Leisch, F. (1998). On the Generation of Correlated Artificial Binary Data. *Adaptive Information Systems and Modelling in Economics and Management Science*, Working Pa(13).

McCullagh, P. (1987). *Tensor Methods in statistics*. Chapman & Hall, New York, USA, first edition.

Nickisch, H. and Rasmussen, C. E. (2008). Approximations for Binary Gaussian Process Classification. *Journal of Machine Learning Research*, 9:2035–2078.

Norris, J. R. (1997). *Markov Chains*. Cambridge University Press, New York, USA, first edit edition.

Oakley, J. E. and O'Hagan, A. (2002). Bayesian Inference for the Uncertainty Distribution of Computer Model Outputs. *Biometrika*, 89(4):769–784.

O'Hagan, A. (1997). Properties of Intrinsic and Fractional Bayes Factors. *Test*, 6(1):101–118.

O'Hagan, A. (2006). Bayesian Analysis of Computer Code Outputs: A Tutorial. *Reliability Engineering and System Safety*, 91.

Paciorek, C. J. and Schervish, M. J. (2006). Spatial Modelling Using a New Class of Nonstationary Covariance Functions. *Environmetrics*, 17(5):483–506.

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R., and Kim, N.-h. (2010). Adaptive Designs of Experiments for Accurate Approximation of Target Regions. *HAL archives-ouvertes*.

Pielou, E. C. (1969). *An Introduction to Mathematical Ecology*. John Wiley and Sons, Inc., United States of America, first edition.

Pielou, E. C. (1984). *The Interpretation of Ecological Data*. John Wiley and Sons, Inc., United States of America, first edition.

Pope, C. A., Gosling, J. P., Barber, S., Johnson, J., Yamaguchi, T., Feingold, G., and Blackwell, P. G. (2018). Modelling Spatial Heterogeneity and Discontinuities using Voronoi Tessellations. *ArXiv*.

Possolo, A. (1986). Estimation of Binary Markov Random Fields. *Technical Report, University of Washington*, (77).

Raftery, A. E. (1985). A Model for High-Order Markov Chains. *Journal of the Royal Statistical Society. Series B (Methodological)*, 47(3):528–539.

Ranjan, P., Bingham, D., and Michailidis, G. (2008). Sequential Experiment Design for Contour Estimation From Complex Computer Codes. *Technometrics*, 50(4):527–541.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*. the MIT Press, Massachusetts Institute of Technology, Cambridge, MA, first edition.

Rhodes, J., Schilling, A., and Silva, P. V. (2017). Random Walks on Semaphore Codes and Delay De Bruijn Semigroups. *ArXiv*.

Rosenthal, J. S. (1995). Convergence Rates for Markov Chains. *SIAM Review*, 37(3):387–405.

Ross, S. M. (2014). *Introduction to Probability Models*. Academic Press, San Diego, California, eleventh e edition.

Rue, H. and Held, L. (2005). *Gaussian Markov Random Fields: Theory and Applications*. Chapman & Hall/CRC, Florida, USA, first edition.

Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989). Design and Analysis of Computer Experiments. *Statistical Science*, 4(4):409–423.

Salter, J. M., Williamson, D. B., Scinocca, J., and Kharin, V. (2019). Uncertainty Quantification for Computer Models with Spatial Output using Calibration-Optimal Bases. *Journal of the American Statistical Association*, 114(528):1800–1814.

Santner, T. J., Williams, B. J., and Notz, W. I. (2003). *The Design and Analysis of Computer Experiments*. Springer-Verlag New York, New York, first edition.

Schmidt, A. M. and O'Hagan, A. (2003). Bayesian Inference for Non-Stationary Spatial Covariance Structure Via Spatial Deformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 65(3):743–758.

Sedgewick, R. and Flajolet, P. (2013). *An Introduction to the Analysis of Algorithms*. Pearson Education, Inc., New Jersey, USA, second edition.

Smith, D. and Smith, M. (2006). Estimation of Binary Markov Random Fields using Markov Chain Monte Carlo. *Journal of Computational and Graphical Statistics*, 15(1):207–227.

Sonin, I. M. (2008). The Decomposition-Separation Theorem for Finite Nonhomogeneous Markov Chains and Related Problems. *IMS Collections: Markov Processes and Related Topics*, 4:1–15.

Strang, G. (2006). *Linear Algebra and its Applications*. Brooks Cole, New York, USA, fourth edition.

Tallis, G. M. (1962). The Use of a Generalized Multinomial Distribution in the Estimation of Correlation in Discrete Data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 24(2):530–534.

Tallis, G. M. (1964). Further Models for Estimating Correlation in Discrete Data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(1):82–85.

Teugels, J. L. (1990). Some Representations of the Multivariate Bernoulli and Binomial Distributions. *Journal of Multivariate Analysis*, 32:256–268.

Turner, B. M. and Van Zandt, T. (2012). A Tutorial on Approximate Bayesian Computation. *Journal of Mathematical Psychology*, 56:69–85.

Vernon, I., Goldstein, M., and Bower, R. G. (2010). Galaxy formation: a Bayesian Uncertainty Analysis. *Bayesian Analysis*, 5(4):619–670.

Voliotis, M., Li, X. F., O'Byrne, K. T., and Tsaneva-Atanasova, K. (2018). A Mathematical Model of the Hypothalamic Network Controlling the Pulsatile Secretion of Reproductive Hormones. *bioRxiv*.

Volodina, V. and Williamson, D. (2020). Diagnostics-Driven Nonstationary Emulators Using Kernel Mixtures. *SIAM/ASA Journal on Uncertainty Quantification*, 8(1):1–26.

Wang, C., Huang, T.-z., and Jia, C. (2013). A Simplified Higher-Order Markov Chain Model. *International Journal of Mathematical and Computational Sciences*, 7(1):174–181.

Welch, W. J., Buck, R. J., Sacks, J., Wynn, H. P., Mitchell, T. J., and Morris, M. D. (1992). Screening, Predicting, and Computer Experiments. *Journal of the*

*American Statistical Association and the American Society for Quality Control*, 34(1):15–25.

Wilf, H. S. (1994). *Generatingfunctionology*. Academic Press, Inc., San Diego, California, second edition.

Wilkinson, R. D. (2008). Approximate Bayesian Computation (ABC) Gives Exact Results Under the Assumption of Model Error. *Biometrika*, 20(10):1–13.

Winkler, G. (1995). *Image Analysis, Random Fields and Dynamic Monte Carlo Methods*. Springer-Verlag, Berlin, Germany, first edition.

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media, Inc., Canada, first edition.

Wunsch, C. (2005). Thermohaline Loops , Stommel Box Models, and the Sandstrom Theorem. *Tellus*, 57(A):84–99.

Zwiernik, P. and Smith, J. Q. (2012). Tree Cumulants and the Geometry of Binary Tree Models. *Bernoulli*, 18(1):290–321.

# Appendix A

# Proofs for Chapter 4 Theorems

## A.1 Theorem 4.2 (Run Length Distribution, $m \geq 3$)

$$
P(\textbf{run length} = n)
$$

$$
= \begin{cases}
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2} & \text{for } n = 1 \\
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} & \text{for } n = 2 : m-1 \\
\qquad \times \left[ \prod_{j=1}^{n-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=n-1}} \right] & \\
\sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} & \\
\qquad \times \left[ \prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)} \right] & \text{for } n \geq m. \\
\qquad \times \left[ \left( p_{2^m-1}^{2^m-1} \right)^{n-m} p_{2^m-1}^{2^m-2} \right] &
\end{cases}
$$

where,

$$
\pi(i) = \sum_{j=0}^{2^m-1} \prod_{k=0}^{m-3} \left[ p_{2^k(j \bmod 2^{m-k})+\sum_{s=1}^{k} 2^{k-s}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]}^{2^{k+1}(j \bmod 2^{m-k-1})+\sum_{s=1}^{k+1} 2^{k-s+1}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]} \right] \pi(j)
$$

*Proof.* Given a sequence of the letters 0 and 1, of length $N$, the sequence can be written in terms of its de Bruijn words of length $m$. Each of these words can be

expressed by the numerical representation of their binary form. Transitioning from the word $i$ to the word $j$ is stated by $p_i^j$.

For a run length of 1's of length $n$, the run starts with a word of the form $*01$ irrespective of the word length, $m$. Hence, $*$ defines a sequence of length $m-2$, and so there are $2^{m-2}$ words of the form $*01$. The law of total probability states:

$$P(\textbf{run length} = n) = \sum_{i=0}^{2^{m-2}-1} P(\textbf{run length} = n|*_i)\pi(*_i),$$

where $\pi(*_i)$ is the marginal probability for the $i$th starting sequence of length $m-2$.

For run lengths of $n = 1$, this corresponds to transitioning from a word of the form $*01$ to a word of the form $*10$. Hence:

$$P(\textbf{run length} = 1|*) = p_{*01}^{*10}.$$

The sequence, $*$, corresponds to all possible length $m-2$ sequences of 0's and 1's which are the binary representations of the numbers $0, 1, ..., 2^{m-2}-1$ (all possible combinations of 0's and 1's). Going from right to left, $*$ starts in the third position representing $2^2$ in the numerical representation. Hence, all starting words of the form $*01$ are given by $4i+1 \ \forall \ i \in (0 : 2^{m-2}-1)$.

From words of the form $*01$, it is then only possible to transition to words of the form $*010$. The sequence, $*$, now represents all possible sequences of length $m-3$, represented by $0, 1, ..., 2^{m-3}-1$. Since the first word is of the form $00\ldots010$, and translating to 2 in the numerical representation, the words are given by $(2^3)i+2 \ \forall \ i \in (0 : 2^{m-3}-1)$. However, since there are only $2^{m-3}$ words to transition to, half of the initial $2^{m-2}$ words transition to the same following word. This forms a repeated cycle of recurring end words that correspond to the starting words in logical order $(1, ..., m)$. The cycle repeats every $2^{m-3}$ and so depends on the word length, $m$. Combining this with the law of total probability gives:

$$P(\textbf{run length} = 1) = \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+2}.$$

For $P(\textbf{run length} = n)$ where $n = 2, ..., m-1$, starting words, $*01$, must transition to a word of the form $*011$ with $p_{*01}^{*011}$. Following this, there will be $n-1$ transitions each adding an extra 1 to the sequence. At each transition, the sequence $*$ will reduce by one letter. The final transition will be to a word of the form $*10$, containing $n$ 1's and stopping the run.

The first transition will have a similar form to the $n = 1$ case with the exception of transitioning to a word of the form $*011$, hence the transition is $p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \forall\ i \in (0 : 2^{m-2} - 1)$. For the remaining transitions, the words will take the form $*01...1$ where the sequence $*$ reduces by 1 each time. The 1's move to the left each time and so the numerical binary number increases by a power of 2 to become $2^{j+3} \forall\ j \in (0 : n-1)$. This is the increase in value for each $i$. As for the $n = 1$ case, a repeated cycle of recurring end words occurs. To begin with, the cycle repeats every $2^{m-3}$, however as the sequence $*$ decreases with increasing transitions, this decreases the total number of possible words and the length of the cycle to give:

$$2^{j+3}(i \bmod 2^{m-j-3}) \qquad \forall\ i \in (0 : 2^{m-2} - 1), j \in (0 : n-1).$$

Also for increasing transitions, the first possible word of the form 0...01...1, gets larger by a power of 2 and so we must add on $2^{j+2} - 1$ for the starting word when $i = 0$.

Finally, the run of 1's is stopped with a word of the form $*10$ and so an indicator is used remove one from the numerical representation of the last transition. Since the end word from the previous transition is the starting word of the next this gives:

$$P(\textbf{run length} = n) = \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \times \left[ \prod_{j=1}^{n-1} p_{2^{j+2}(i \bmod 2^{m-j-2})+(2^{j+1}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)-\mathbb{1}_{j=n-1}} \right],$$

for $n = 2, ..., m-1$.

For run lengths $n \geq m$, the initial $m-1$ transitions will be identical to the $n < m$ cases where the indicator is removed. The remaining $n - m$ transition probabilities

are of the form $p_{11\ldots1}^{11\ldots1} = p_{2^{m-1}}^{2^{m-1}}$. The last word is of the form $11...10 = 2^m - 2$, hence:

$$P(\textbf{run length} = n) = \sum_{i=0}^{2^{m-2}-1} \pi(i)\, p_{4i+1}^{2^3(i \bmod 2^{m-3})+3} \left[\prod_{j=1}^{m-2} p_{2^{j+2}(i \bmod 2^{m-j-3})+(2^{j+2}-1)}^{2^{j+3}(i \bmod 2^{m-j-3})+(2^{j+2}-1)}\right] \times \left[\left(p_{2^m-1}^{2^m-1}\right)^{n-m} p_{2^m-1}^{2^m-2}\right],$$

for $n \geq m$.

For $\pi(i) \quad \forall i \in (0 : 2^{m-2} - 1)$, the law of total probability states:

$$\pi(i) = \sum_{j=0}^{2^{m-2}-1} P(i|j)\pi(j),$$

where the numerical representation of the binary words are used. Since the run must start with a word of the form $*01$, $\pi(i)$ represents all possible sequences, $*$ of length $m-2$. $\pi$ is defined in terms of length $m$ words and associated transition probabilities. Finding the conditional probabilities involves $m-2$ transition probabilities for each $j \in (0 : 2^m - 1)$ since there are $2^m$ words in an $m$ length de Bruijn graph.

First consider the case when $i = 0$ for any word length $m$ (in the numerical representation of the binary word, 0...0). The first word for each $j$, is always $j$ itself. For each following transition, for each $j$, the letters shift to the left, and a 0 is added to the end of the word. Hence, the word becomes an extra factor of 2 larger at each transition i.e. $2^k \times j$ for each $k \in (0 : m-3)$. However, since the word transitions to a word of the form $*0...0$ each time, where $*$ reduces in length by one, some words can transition to the same word, creating a cycle on $j$. Initially all $2^m$ words are unique, but the cycle reduces with each transition (as $*$ becomes smaller), so the words become: $2^k(j \bmod 2^{m-k})$.

If $i \in (0 : 2^{m-2} - 1)$, the initial pattern is the same as for $i = 0$, with different contributions to each word for each transition. 1's must be added to the base case ($i = 0$) when they turn up in the transitions for any $i$. For all $2^{m-2}$ possible binary sequences of length $m-2$, the 1's appear in cycles depending on which column they lie in. From the left entry, the first $2^{m-3}$ words begin with a 0 and the last $2^{m-3}$ words begin with a 1. The next column repeats 0's and 1's every

$2^{m-4}$ words. This continues until the last column alternates between 0's and 1's for each $i$. This reduces to $(\frac{1}{2^{m-s-2}}(i - (i \bmod 2^{m-s-2})))\bmod 2$, where $s \in (1 : k)$ for each transition. Since the current 1's increase by a power of two for each transition, a summation is formed to make sure each is included. Hence the words become: $\sum_{s=1}^{k} 2^{k-s}[(\frac{1}{2^{m-s-2}}(i - (i \bmod 2^{m-s-2})))\bmod 2]$.

Since the end word of the current transition is the start word of the next transition, this gives the following:

$$\pi(i) = \sum_{j=0}^{2^m-1} \prod_{k=0}^{m-3} \left[ p_{2^k(j \bmod 2^{m-k}) + \sum_{s=1}^{k} 2^{k-s}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]}^{2^{k+1}(j \bmod 2^{m-k-1}) + \sum_{s=1}^{k+1} 2^{k-s+1}[(\frac{1}{2^{m-s-2}}(i-(i \bmod 2^{m-s-2})))\bmod 2]} \right] \pi(j)$$

□

## A.2 Expected Run Length for $m = 2$ Obtained from Moment Generating Function

$$E\left[\textbf{run length}\right] = p_{01}^{10} + \frac{p_{01}^{11}\left(1-(p_{11}^{10})^2\right)}{p_{11}^{11}p_{11}^{10}}.$$

*Proof.*

$$G(x) = \frac{\left(p_{01}^{11}p_{11}^{10} - p_{01}^{10}p_{11}^{11}\right)e^{2x} + p_{01}^{10}e^{x}}{1 - p_{11}^{11}e^{x}}$$

$$G'(x) = \frac{\left[2\left(p_{01}^{11}p_{11}^{10} - p_{01}^{10}p_{11}^{11}\right)e^{2x} + p_{01}^{10}e^{x}\right]\left(1 - p_{11}^{11}e^{x}\right)}{\left(1 - p_{11}^{11}e^{x}\right)^2} - \frac{\left[\left(p_{01}^{11}p_{11}^{10} - p_{01}^{10}p_{11}^{11}\right)e^{2x} + p_{01}^{10}e^{x}\right]\left(p_{11}^{11}e^{x}\right)}{\left(1 - p_{11}^{11}e^{x}\right)^2}$$

$$\begin{aligned}
G'(0) &= \frac{\left[2\left(p_{01}^{11}p_{11}^{10} - p_{01}^{10}p_{11}^{11}\right) + p_{01}^{10}\right]\left(1 - p_{11}^{11}\right) - \left[\left(p_{01}^{11}p_{11}^{10} - p_{01}^{10}p_{11}^{11}\right) + p_{01}^{10}\right]\left(p_{11}^{11}\right)}{\left(1 - p_{11}^{11}\right)^2} \\
&= \frac{2p_{01}^{11}p_{11}^{10} - 2p_{01}^{10}p_{11}^{11} + p_{01}^{10} - p_{01}^{11}p_{11}^{10}p_{11}^{11} + p_{01}^{10}\left(p_{11}^{11}\right)^2}{\left(1 - p_{11}^{11}\right)^2} \\
&= \frac{p_{01}^{10}\left[\left(p_{11}^{11}\right)^2 - 2p_{11}^{11} + 1\right]}{\left(1 - p_{11}^{11}\right)^2} + \frac{2p_{01}^{11}p_{11}^{10} - p_{01}^{11}p_{11}^{10}p_{11}^{11}}{\left(1 - p_{11}^{11}\right)^2} \\
&= p_{01}^{10} + \frac{2p_{01}^{11}p_{11}^{10} - p_{01}^{11}p_{11}^{10}p_{11}^{11}}{\left(p_{11}^{10}\right)^2} \\
&\text{since } \left(1 - p_{11}^{11}\right)^2 = \left(p_{11}^{11}\right)^2 - 2p_{11}^{11} + 1 \text{ and } p_{11}^{10} + p_{11}^{11} = 1 \\
&= p_{01}^{10} + \frac{2p_{01}^{11} - p_{01}^{11}p_{11}^{11}}{p_{11}^{10}} \\
&= p_{01}^{10} + \frac{p_{01}^{11}}{p_{11}^{10}p_{11}^{11}}\left[2p_{11}^{11} - \left(p_{11}^{11}\right)^2\right] \\
&= p_{01}^{10} + \frac{p_{01}^{11}}{p_{11}^{10}p_{11}^{11}}\left[2\left(1 - p_{11}^{10}\right) - \left(1 - p_{11}^{10}\right)^2\right] \\
&\text{since } p_{11}^{10} + p_{11}^{11} = 1 \\
&= p_{01}^{10} + \frac{p_{01}^{11}\left(1 - (p_{11}^{10})^2\right)}{p_{11}^{11}p_{11}^{10}} \\
&= E\left[\textbf{run length}\right] \text{ from Theorem 4.3.}
\end{aligned}$$

□

## A.3 Theorem 4.18 (Marginal Likelihood, $m \geq 1$)

$$\begin{aligned}\mathcal{L}(\text{seq}) &= \prod_{i=0}^{2^m-1} \pi(i)^{n_i} \\ &= \prod_{i=0}^{2^m-2} \pi(i)^{n_i} \left[1 - \left(\sum_{j=0}^{2^m-2} \pi(j)\right)\right]^{N-\left(\sum_{j=0}^{2^m-2} n_j\right)}\end{aligned}$$

for $N = \sum_{j=0}^{2^m-1} n_j$ and $\sum_{j=0}^{2^m-1} \pi(j) = 1$

*Proof.* Given a sequence of letters of length $N + m - 1$, the sequence can be split up into its associated words of predefined length $m \geq 1$, where $\pi(i)$ gives the probability of obtaining the word $i$ and $N$ is the total number of words. The joint distribution is given by the product of the respective $\pi$'s for each word and like terms are collected. There are $2^m$ words for an $m$ length de Bruijn process, hence:

$$\mathcal{L}(n) = \prod_{i=0}^{2^m-1} \pi(i)^{n_i}$$

Since the sum of all $n_i$ gives the total number of words, $N$, and the sum of the marginal probabilities of the words equals one, this gives the following:

$$\mathcal{L}(n) = \prod_{i=0}^{2^m-2} \pi(i)^{n_i} \left[1 - \left(\sum_{j=0}^{2^m-2} \pi(j)\right)\right]^{N-\left(\sum_{j=0}^{2^m-2} n_j\right)}$$

□

## A.4 Theorem 4.20 (Transition Likelihood, $m \geq 1$)

$$\mathcal{L}(seq|p) = \prod_{i=0}^{2^m-1} \left(1 - p_i^{(2i+1) \bmod 2^m}\right)^{n_i^{((2i+1) \bmod 2^m)-1}} \left(p_i^{(2i+1) \bmod 2^m}\right)^{n_i^{((2i+1) \bmod 2^m)}}.$$

*Proof.* Assume a sequence of letters, $L = l_1, l_2, ..., l_n$, where $l_i \in [0, 1]$ and the ordering is fixed. This sequence can be written in terms of its de Bruijn words such that $L = w_1, w_2, ..., w_{n-1}$, where $w_i$ are the de Bruijn words of length $m$. Consider the joint distribution of this sequence. Starting from $w_1$, the probability of transitioning to the next word is $p_{w_1}^{w_2}$. The probability of transitioning to the next following word is, $p_{w_2}^{w_3}$. This continues until the transition $p_{w_{n-2}}^{w_{n-1}}$ which gives the joint distribution, $\mathcal{L}(L|p) = p_{w_1}^{w_2} \times p_{w_2}^{w_3} \times ... \times p_{w_{n-2}}^{w_{n-1}}$. Like terms are collected for each possible transition probability.

There are $2^{m+1}$ possible transition probabilities since each word can be followed by either a 0 or a 1. Since rows in the transition matrix sum to one, the transition likelihood can be expressed in terms of $2^m$ parameters. These are all words of the form $*1$, which end in the letter 1 and so are expressed as $2i + 1$ for $i \in (0 : 2^m - 1)$ (using the numerical representation of the binary words). Since all possible words can transition to a word of the form $*1$, where $*$ is of length $m - 1$, two words each transition to the same end word and a cycle is formed. This occurs every $2^m$ transitions, hence:

$$\mathcal{L}(L|p) = \prod_{i=0}^{2^m-1} \left(1 - p_i^{((2i+1) \bmod 2^m)}\right)^{n_i^{((2i+1) \bmod 2^m)-1}} \left(p_i^{((2i+1) \bmod 2^m)}\right)^{n_i^{((2i+1) \bmod 2^m)}}.$$

□

## A.5 Theorem 4.21 (Estimation of De Bruijn Word length by Bayes' factors, $m \geq 1$)

Consider a sequence, $S$, of 0's and 1's which was generated under one of two hypotheses. The first is a de Bruijn process with word length $m_1$ and the second is a de Bruijn process with word length $m_2$. The Bayes' factor ratio is as follows:

$$B_{1,2} = \frac{P(S|m_1)}{P(S|m_2)}$$

where,

$$\begin{aligned} P(S|m_k) &= \int P(S|p, m_k)P(p|m_k)dp \\ &= \prod_{i=0}^{2^{m_k}-1} \frac{\Gamma(n_i^{((2i+1) \bmod 2^{m_k})-1} + \beta_{i+1})\Gamma(n_i^{((2i+1) \bmod 2^{m_k})}) + \alpha_{i+1})}{\Gamma(n_i^{((2i+1) \bmod 2^{m_k})-1} + n_i^{((2i+1) \bmod 2^{m_k})} + \beta_{i+1} + \alpha_{i+1})}, \end{aligned}$$

for $k \in \{1, 2\}$. When values of $B_{1,2}$ are large, we have more evidence to reject the first hypothesis with word length $m_1$ in favour of the second hypothesis with word length $m_2$.

*Proof.* The equation:

$$\begin{aligned} P(p|S) &= \frac{\mathcal{L}(S|p, m)P(p|m)}{P(S)} \\ &= \frac{\mathcal{L}(S|p, m)P(p|m)}{\int \mathcal{L}(S|p, m)P(p|m)dp} \end{aligned}$$

is taken from Bayes' theorem where the likelihood is given in Theorem 4.20 and the prior is defined to be a product of beta densities:

$$P(p|m) = \prod_{i=0}^{2^m-1} \frac{\Gamma(\alpha_{i+1} + \beta_{i+1})}{\Gamma(\alpha_{i+1})\Gamma(\beta_{i+1})} p^{\alpha_{i+1}-1}(1-p)^{\beta_{i+1}-1},$$

for unknown parameters $\alpha$ and $\beta$. Substituting this in gives:

$$
\begin{aligned}
P(p|S) &\propto \mathcal{L}(S|p,m)P(p|m) \\
&= \prod_{i=0}^{2^m-1} \Bigg[ \left(1 - p_i^{((2i+1) \bmod 2^m)}\right)^{n_i^{((2i+1) \bmod 2^m)-1}} \left(p_i^{((2i+1) \bmod 2^m)}\right)^{n_i^{((2i+1) \bmod 2^m)}} \\
&\qquad \times \left(1 - p_i^{((2i+1) \bmod 2^m)}\right)^{\beta_{i+1}-1} \left(p_i^{((2i+1) \bmod 2^m)}\right)^{\alpha_{i+1}-1} \Bigg] \\
&= \prod_{i=0}^{2^m-1} (1 - p_i^{((2i+1) \bmod 2^m)})^{n_i^{((2i+1) \bmod 2^m)-1}+\beta_{i+1}-1} \\
&\qquad \times (p_i^{((2i+1) \bmod 2^m)})^{n_i^{((2i+1) \bmod 2^m)}+\alpha_{i+1}-1}.
\end{aligned}
$$

Both the prior, $P(p|m)$, and the posterior, $P(p|S)$, take the form of a product of beta densities, hence there is a conjugate relationship and the following is true:

$$
\int P(S|p,m)P(p|m)dp = \prod_{i=0}^{2^m-1} \frac{\Gamma(n_i^{((2i+1) \bmod 2^m)-1} + \beta_{i+1})\Gamma(n_i^{((2i+1) \bmod 2^m)}) + \alpha_{i+1})}{\Gamma(n_i^{((2i+1) \bmod 2^m)-1} + n_i^{((2i+1) \bmod 2^m)} + \beta_{i+1} + \alpha_{i+1})}.
$$

□

# Appendix B

# Proofs for Chapter 6 Theorems

## B.1 Theorem 6.2 (Run Length Distribution, $m \geq 2$)

$$
P(\textbf{run length} = n)
$$

$$
= \begin{cases}
\sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} & \text{for } n = 1 \\
\sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) & \text{for } n = 2:2m \\
\quad \times \prod_{k=0}^{n-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] : \right. & \\
\quad \left. \sum_{t=0}^{\min(n-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{n-k-1}} \left[j - (j \bmod 2^{n-k-1})\right] \right\} & \\
\sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) & \text{for } n \geq 2m+1 \\
\quad \times \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] : \right. & \\
\quad \left. \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[j - (j \bmod 2^{2m-k-1})\right] \right\} & \\
\quad \times \left( p^1_{2^m-1:2^m-1} \right)^{n-2m} &
\end{cases}
$$

*Proof.* Given a sequence of the letters 0 and 1, of length, $N$, the sequence can be written in terms of its de Bruijn words of length $m$. Each of these words can be expressed by the numerical representation of their binary form. The probability of getting the letter 1 from the word $i:j$ is given by $p^1_{i:j}$.

For a run length of 1's of length $n$, the run is bounded by two words of the form $*0:...$ and $...:0*'$ symmetrically. The sequences represented by ... are dependent on the run length, and both $*$ and $*'$ are any sequence of 0's and 1's of length $m-1$. Hence, there are $2^{m-1}$ words of either the form $*0:...$ or $...:0*'$, which gives $2^{2(m-1)}$ combinations in total. The law of total probability states:

$$P(\textbf{run length} = n) = \sum_{i=0}^{2^{2(m-1)}-1} P(\textbf{run length} = n | * \_ *'_i)\pi(* \_ *'_i),$$

where $\pi(* \_ *'_i)$ is the marginal probability for the $i$th combination of starting and end sequences of length $m-1$ each.

For run lengths of $n = 1$, this corresponds to transitioning from a word of the form $*0:0*'$ to the letter 1. Hence:

$$P(\textbf{run length} = 1 | * \_ *') = p^{*1}_{*0:0*'}.$$

The sequence, $*$, corresponds to all possible length $m-1$ sequences of 0's and 1's, hence $*0$ corresponds to all even numbers in the numerical representation of the binary sequences, $2i \ \forall \ i \in (0 : 2^{m-1} - 1)$. The sequence, $*'$ also corresponds to all possible length $m-1$ sequences, and so $0*'$ corresponds to $j \ \forall \ j \in (0 : 2^{m-1} - 1)$. Since each possible starting sequence, $*$, for a run can finish with each ending sequence, $*'$, this gives:

$$P(\textbf{run length} = 1) = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i \_ j) p^1_{2i:j},$$

where $\pi(i \_ j)$ is the marginal probability of getting the $i$th possible starting sequence represented by $*$ and the $j$th possible ending sequence represented by $*'$.

For $P(\textbf{run length} = n)$ where $n = 2, ..., 2m$, there will be $n$ conditional word probabilities for each of the 1's in the run. Each non-directional word is of the

form $w_1 : w_2$, where each of these sequences, $w_k$, is of length $m$. Due to the non-directionality, that patterns that occur with $w_1$ will also symmetrically occur with $w_2$. Considering the left and right sides of the word separately, $w_1$ will start off as a sequence of the form $*0$ and go to a sequence of the form $*01$. This continues, where at each time, the sequence $*$ reduces in length by one and an extra letter 1 is added to the end. If $*$ is shifting to the left and reducing in length each time, the numerical binary number represented by this will increase by a power of two each time to become $2^{k+1} \forall\ k \in (1 : n-2)$. The baseline is $2^2 = 4$ since the sequence will be of the form $*01$ where $*$ starts in the third position from the right. As $*$ reduces in length with $k$, there are less options for what the sequence can be. Hence, a cycle forms of repeated start sequences. Initially this cycle is of length $2^{m-2}$, which reduces by a factor of 2 with each transition until the cycle is of length $2^{m-n+1}$. Therefore, $w_1$ becomes $2^{k+1}(j \bmod 2^{m-k-1}) \forall\ k \in (0 : n-1)$. For all run lengths larger than $m$, this expression reduces to 0 since $w_1$ consists entirely of 1's.

Whilst this happens, an extra 1 is added to the sequence at each transition. Hence the corresponding number must be added to the result to represent these 1's. This starts from one and increases by a power of 2 at each transition until $w_1$ is of the form 1...1. This is given by $2^{\min(k,m)} - 1 \forall\ k \in (0 : n-1)$. The minimum value between $k$ and $m$ is taken since when $w_1$ consists entirely of 1's, its maximum value is $2^m - 1$.

Putting these together gives: $w_1 = 2^{\min(k,m)} - 1 + \left[2^{k+1}(j \bmod 2^{m-k-1})\right]$ for $k \in (0 : n-1)$.

Now consider the right side of the word, $w_2$. The patterns emerging in $w_2$ will be the reverse to the patterns in $w_1$ due to the non-directionality of the form of the words. $w_2$ will start off as a sequence of either the form 1...1 or $1...10*'$ depending on the run length $n$. As the number of transitions increase, the sequence $*$ increases in length by one each time and a 1 is removed from the initial run. This continues until the last transition, and $w_2$ is of the form $0*'$, where $*'$ is any sequence of 0's and 1's of length $m-1$. Accounting for the initial run of 1's, $\sum_{t=0}^{\min(n-k-1,m-1)} 2^{m-t-1}$ adds in the numeric value for each of the 1's starting from the left with $2^{m-1}$. Taking the

minimum value between $n-k-2$ and $m-1$ for the summation in necessary for when $w_2$ consists of entirely 1's and has maximum value $2^{m-1}+2^{m-2}+...+2^0$.

As the number of 1's in $w_2$ decreases, the length of the sequence $*$ increases. Depending on the current value of $k$, $*'$ is of length $m-n+k$ which gives $2^{m-n+k}$ different sequences of 0's and 1's. Since $j \in (0:2^{m-1}-1)$ and there must be all combinations of $w_1$ with $w_2$, each sequence $*$ will occur $2^{m-k}$ times. This is given by $\frac{1}{2^{n-k-1}}\left[j-(j \bmod 2^{n-k-1})\right]$, which gives the sequence $0...01...1...2^{m-n+k}...2^{m-n+k}$. When $w_2$ consists entirely of 1's, $n-k-1 \geq m$ and just 0's are produced.

Putting these steps together gives: $w_2 = \sum_{t=0}^{\min(n-k-2,m-1)}\{2^{m-t-1}\} + \frac{1}{2^{n-k-1}}\left[j-(j \bmod 2^{n-k-1})\right]$ for $k \in (0:n-1)$.

Therefore, for $n=2,...,2m$ the run length distribution is given by:

$$P(\textbf{run length}=n) = \sum_{i=0}^{2^{m-1}-1}\sum_{j=0}^{2^{m-1}-1}\pi(i\_j) \times \prod_{k=0}^{n-1}\left\{p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]:\ \sum_{t=0}^{\min(n-k-2,m-1)}\{2^{m-t-1}\}+\frac{1}{2^{n-k-1}}\left[j-(j \bmod 2^{n-k-1})\right]}\right\}$$

For run lengths $n \geq 2m+1$, the run-in periods occurring for both of the end $m$ transitions will be identical to the $n \leq 2m$ cases. For longer run lengths than $2m$, the following $n-2m$ transitions are of the form $p^1_{1...1:1...1}$ where both $w_1$ and $w_2$ are sequences of 1's of length $m$. This is represented by $p^1_{2^m-1:2^m-1}$.

Therefore, for $n \geq 2m+1$ the run length distribution is given by:

$$P(\textbf{run length}=n) = \sum_{i=0}^{2^{m-1}-1}\sum_{j=0}^{2^{m-1}-1}\pi(i\_j) \times \prod_{k=0}^{n-1}\left\{p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]:\ \sum_{t=0}^{\min(n-k-2,m-1)}\{2^{m-t-1}\}+\frac{1}{2^{n-k-1}}\left[j-(j \bmod 2^{n-k-1})\right]}\right\} \times \left(p^1_{2^m-1,2^m-1}\right)^{n-2m}$$

□

## B.2 Theorem 6.4 (Expected Run Length, $m \geq 2$)

$$
\begin{aligned}
&E\left[\textbf{run length}\right] \\
&= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
&+ \sum_{l=2}^{2m} l \left[ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{l-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] : \right.\right. \\
&\qquad \left.\left. \sum_{t=0}^{\min(l-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{l-k-1}} \left[ j - (j \bmod 2^{l-k-1}) \right] \right\} \right] \\
&+ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] : \right. \\
&\qquad \left. \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j - (j \bmod 2^{2m-k-1}) \right] \right\} \\
&\quad \times \left\{ \left( \frac{1}{p^1_{2^m-1:2^m-1}} \right)^{2m} \left[ \frac{p^1_{2^m-1:2^m-1}}{(1 - p^1_{2^m-1:2^m-1})^2} - \sum_{s=0}^{2m} s (p^1_{2^m-1:2^m-1})^s \right] \right\}
\end{aligned}
$$

*Proof.*

$$
\begin{aligned}
&E\left[\textbf{run length}\right] \\
&= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
&+ \sum_{l=2}^{2m} l \left[ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{l-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] : \right.\right. \\
&\qquad \left.\left. \sum_{t=0}^{\min(l-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{l-k-1}} \left[ j - (j \bmod 2^{l-k-1}) \right] \right\} \right] \\
&+ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] : \right. \\
&\qquad \left. \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j - (j \bmod 2^{2m-k-1}) \right] \right\} \\
&\quad \times \left\{ \sum_{n=2m+1}^{\infty} n (p^1_{2^m-1:2^m-1})^{n-2m} \right\}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
&+ \sum_{l=2}^{2m} l \left[ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{l-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right.\right. \\
&\qquad \left.\left. \sum_{t=0}^{\min(l-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{l-k-1}} \left[ j - (j \bmod 2^{l-k-1}) \right] \right\} \right] \\
&+ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right. \\
&\qquad \left. \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j - (j \bmod 2^{2m-k-1}) \right] \right\} \\
&\quad \times \left\{ \left( \frac{1}{p^1_{2^m-1:2^m-1}} \right)^{2m} \left[ \sum_{n=0}^{\infty} n (p^1_{2^m-1:2^m-1})^n - \sum_{s=0}^{2m} s (p^1_{2^m-1:2^m-1})^s \right] \right\}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
&+ \sum_{l=2}^{2m} l \left[ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{l-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right.\right. \\
&\qquad \left.\left. \sum_{t=0}^{\min(l-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{l-k-1}} \left[ j - (j \bmod 2^{l-k-1}) \right] \right\} \right] \\
&+ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right. \\
&\qquad \left. \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j - (j \bmod 2^{2m-k-1}) \right] \right\} \\
&\quad \times \left\{ \left( \frac{1}{p^1_{2^m-1:2^m-1}} \right)^{2m} \left[ \frac{p^1_{2^m-1:2^m-1}}{(1 - p^1_{2^m-1:2^m-1})^2} - \sum_{s=0}^{2m} s (p^1_{2^m-1:2^m-1})^s \right] \right\}
\end{aligned}
$$

for $|p^1_{2^m-1:2^m-1}| < 1$

since $\sum_{n=0}^{\infty} n p^n = \frac{p}{(p-1)^2}$ providing $|p| < 1$ □

## B.3 Theorem 6.6 (Squared Expectation of Run Length, $m \geq 2$)

$$
\begin{aligned}
&E\left[\textbf{run length}^2\right] \\
&= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
&+ \sum_{l=2}^{2m} l^2 \left[ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{l-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right.\right. \\
&\qquad \left.\left. \sum_{t=0}^{\min(l-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{l-k-1}} \left[ j - (j \bmod 2^{l-k-1}) \right] \right\} \right] \\
&+ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right. \\
&\qquad \left. \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j - (j \bmod 2^{2m-k-1}) \right] \right\} \\
&\quad \times \left\{ \left( \frac{1}{p^1_{2^m-1:2^m-1}} \right)^{2m} \left[ \frac{p^1_{2^m-1:2^m-1}(p^1_{2^m-1:2^m-1}+1)}{(p^1_{2^m-1:2^m-1}-1)^3} + \sum_{s=0}^{2m} s^2 (p^1_{2^m-1:2^m-1})^s \right] \right\}
\end{aligned}
$$

for $p^1_{2^m-1:2^m-1} < 1$

*Proof.*

$$
\begin{aligned}
&E\left[\textbf{run length}^2\right] \\
&= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
&+ \sum_{l=2}^{2m} l^2 \left[ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{l-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right.\right. \\
&\qquad \left.\left. \sum_{t=0}^{\min(l-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{l-k-1}} \left[ j - (j \bmod 2^{l-k-1}) \right] \right\} \right] \\
&+ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right. \\
&\qquad \left. \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j - (j \bmod 2^{2m-k-1}) \right] \right\} \\
&\quad \times \left\{ \sum_{n=2m+1}^{\infty} n^2 (p^1_{2^m-1:2^m-1})^{n-2m} \right\}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
&+ \sum_{l=2}^{2m} l^2 \left[ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{l-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right.\right. \\
&\qquad \left.\left. \sum_{t=0}^{\min(l-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{l-k-1}} \left[ j - (j \bmod 2^{l-k-1}) \right] \right\} \right] \\
&+ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right. \\
&\qquad \left. \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j - (j \bmod 2^{2m-k-1}) \right] \right\} \\
&\quad \times \left\{ \left( \frac{1}{p^1_{2^m-1:2^m-1}} \right)^{2m} \left[ \sum_{n=0}^{\infty} n^2 (p^1_{2^m-1:2^m-1})^n - \sum_{s=0}^{2m} s^2 (p^1_{2^m-1:2^m-1})^s \right] \right\}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
&+ \sum_{l=2}^{2m} l^2 \left[ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{l-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right.\right. \\
&\qquad \left.\left. \sum_{t=0}^{\min(l-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{l-k-1}} \left[ j - (j \bmod 2^{l-k-1}) \right] \right\} \right] \\
&- \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]} : \right. \\
&\qquad \left. \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j - (j \bmod 2^{2m-k-1}) \right] \right\} \\
&\quad \times \left\{ \left( \frac{1}{p^1_{2^m-1:2^m-1}} \right)^{2m} \left[ \frac{p^1_{2^m-1:2^m-1}(p^1_{2^m-1:2^m-1}+1)}{(p^1_{2^m-1:2^m-1}-1)^3} + \sum_{s=0}^{2m} s^2 (p^1_{2^m-1:2^m-1})^s \right] \right\}
\end{aligned}
$$

for $p^1_{2^m-1:2^m-1} < 1$

since $\sum_{n=0}^{\infty} n^2 p^n = -\frac{p(p+1)}{(p-1)^3}$ providing $|p| < 1$ □

## B.4 Theorem 6.8 (Run Length Probability Generating Function, $m \geq 2$)

$$G(x) = \sum_{s=0}^{2m} a_s x^s + \frac{p^1_{2^m-1:2^m-1} a_{2m} x^{2m+1}}{1 - p^1_{2^m-1:2^m-1} x}$$

where,

$$a_1 = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j}$$

$$a_2 = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \left\{ p^1_{2i\,:\,2^{m-1}+\frac{1}{2}[j-(j \bmod 2)]} \right\} \times \left\{ p^1_{1+4[i \bmod 2^{m-2}]\,:\,j} \right\}$$

$$\vdots$$

$$a_{2m} = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \times \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]\,:\, \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}}\left[j-(j \bmod 2^{2m-k-1})\right]} \right\}$$

*Proof.*

$$P(\textbf{run length} = n) = \begin{cases} \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} & \text{for } n = 1 \\ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) & \text{for } n = 2 : 2m \\ \quad \times \prod_{k=0}^{n-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]\,:\, \sum_{t=0}^{\min(n-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{n-k-1}}\left[j-(j \bmod 2^{n-k-1})\right]} \right\} & \\ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) & \text{for } n \geq 2m+1 \\ \quad \times \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]\,:\, \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}}\left[j-(j \bmod 2^{2m-k-1})\right]} \right\} & \\ \quad \times \left( p^1_{2^m-1:2^m-1} \right)^{n-2m} & \end{cases}$$

The recurrence relationship has the following form:

$$
\begin{aligned}
a_0 &= 0 \\
a_1 &= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
a_2 &= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \left\{ p^1_{2i\,:\,2^{m-1}+\frac{1}{2}[j-(j \bmod 2)]} \right\} \times \left\{ p^1_{1+4[i \bmod 2^{m-2}]\,:\,j} \right\} \\
&\vdots \\
a_{n+1} &= p^1_{2^m-1:2^m-1} a_n \qquad \text{for} \quad n \geq 2m
\end{aligned}
$$

This is solved to find the generating function $G = \sum_{n\geq 0} a_n x^n$. Then multiplying by $x^n$ and summing over $n$ gives the following:

$$
\begin{aligned}
\sum_{n\geq 2m} a_{n+1} x^n &= \sum_{n\geq 2m} p^1_{2^m-1:2^m-1} a_n x^n \\
\left(a_{2m+1}x^{2m} + a_{2m+2}x^{2m+1}\right. & \\
\left. + a_{2m+3}x^{2m+2} + \ldots\right) &= p^1_{2^m-1:2^m-1} \left( \sum_{n\geq 0} a_n x^n - a_0 - a_1 x - \ldots \right. \\
& \qquad \left. - a_{2m-1}x^{2m-1} \right) \\
\frac{1}{x}\Big[ \left(a_0 + a_1 x + a_2 x^2 + \ldots\right) & \\
-a_0 - a_1 x - \ldots - a_{2m} x^{2m} \Big] &= p^1_{2^m-1:2^m-1} \left( G - \sum_{s=0}^{2m-1} a_s x^s \right) \\
\frac{G - \sum_{t=0}^{2m} a_t x^t}{x} &= p^1_{2^m-1:2^m-1} \left( G - \sum_{s=0}^{2m-1} a_s x^s \right) \\
G - \sum_{t=0}^{2m} a_t x^t &= x p^1_{2^m-1:2^m-1} G - x p^1_{2^m-1:2^m-1} \sum_{s=0}^{2m-1} a_s x^s \\
G\left(x p^1_{2^m-1:2^m-1} - 1\right) &= x p^1_{2^m-1:2^m-1} \sum_{s=0}^{2m} a_s x^s - \sum_{t=0}^{2m} a_t x^t - x p^1_{2^m-1:2^m-1} a_{2m} x^{2m} \\
G &= \frac{\left(x p^1_{2^m-1:2^m-1} - 1\right)\left(\sum_{s=0}^{2m} a_s x^s\right) - x p^1_{2^m-1:2^m-1} a_{2m} x^{2m}}{p^1_{2^m-1:2^m-1} x - 1} \\
G &= \sum_{s=0}^{2m} a_s x^s + \frac{p^1_{2^m-1:2^m-1} a_{2m} x^{2m+1}}{1 - p^1_{2^m-1:2^m-1} x}
\end{aligned}
$$

where,

$$
\begin{aligned}
a_1 &= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
a_2 &= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \left\{ p^1_{2i \,:\, 2^{m-1}+\frac{1}{2}[j-(j \bmod 2)]} \right\} \times \left\{ p^1_{1+4[i \bmod 2^{m-2}] \,:\, j} \right\} \\
&\vdots \\
a_{2m} &= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \\
&\qquad \times \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right] \,:\, \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j-(j \bmod 2^{2m-k-1}) \right]} \right\}
\end{aligned}
$$

□

## B.5 Theorem 6.10 (Run Length Moment Generating Function, $m \geq 3$)

$$G(x) = \sum_{s=0}^{2m} a_s e^{sx} + \frac{p^1_{2^m-1:2^m-1} a_{2m} e^{(2m+1)x}}{1 - p^1_{2^m-1:2^m-1} e^x}$$

where,

$$a_1 = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j}$$

$$a_2 = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \left\{ p^1_{2i\,:\,2^{m-1}+\frac{1}{2}[j-(j \bmod 2)]} \right\} \times \left\{ p^1_{1+4[i \bmod 2^{m-2}]\,:\,j} \right\}$$

$$\vdots$$

$$a_{2m} = \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \times \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]\,:\, \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}}\left[j-(j \bmod 2^{2m-k-1})\right]} \right\}$$

*Proof.*

$$P(\textbf{run length} = n) = \begin{cases} \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} & \text{for } n = 1 \\ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) & \text{for } n = 2 : 2m \\ \quad \times \prod_{k=0}^{n-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]\,:\, \sum_{t=0}^{\min(n-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{n-k-1}}\left[j-(j \bmod 2^{n-k-1})\right]} \right\} & \\ \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) & \text{for } n \geq 2m+1 \\ \quad \times \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]\,:\, \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}}\left[j-(j \bmod 2^{2m-k-1})\right]} \right\} & \\ \quad \times \left( p^1_{2^m-1:2^m-1} \right)^{n-2m} & \end{cases}$$

The recurrence relationship has the following form:

$$
\begin{aligned}
a_0 &= 0 \\
a_1 &= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
a_2 &= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \left\{ p^1_{2i\,:\,2^{m-1}+\frac{1}{2}[j-(j \bmod 2)]} \right\} \times \left\{ p^1_{1+4[i \bmod 2^{m-2}]\,:\,j} \right\} \\
&\vdots \\
a_{n+1} &= p^1_{2^m-1:2^m-1} a_n \qquad \text{for} \quad n \geq 2m
\end{aligned}
$$

This is solved to find the generating function $G = \sum_{n\geq 0} a_n e^{nx}$. Then, multiplying by $e^{nx}$ and summing over $n$ gives the following:

$$
\begin{aligned}
\sum_{n\geq 2m} a_{n+1} e^{nx} &= \sum_{n\geq 2m} p^1_{2^m-1:2^m-1} a_n e^{nx} \\
\left(a_{2m+1} e^{2mx} + a_{2m+2} e^{(2m+1)x}\right. & \\
\left. + a_{2m+3} e^{(2m+2)x} + \ldots \right) &= p^1_{2^m-1:2^m-1} \left( \sum_{n\geq 0} a_n e^{nx} - a_0 - a_1 e^x - \ldots \right. \\
&\qquad \left. - a_{2m-1} e^{(2m-1)x} \right) \\
\frac{1}{e^x} \Big[ \left(a_0 + a_1 e^x + a_2 e^{2x} + \ldots\right) & \\
-a_0 - a_1 e^x - \ldots - a_{2m} e^{2mx} \Big] &= p^1_{2^m-1:2^m-1} \left( G - \sum_{s=0}^{2m-1} a_s e^{sx} \right) \\
\frac{G - \sum_{t=0}^{2m} a_t e^{tx}}{e^x} &= p^1_{2^m-1:2^m-1} \left( G - \sum_{s=0}^{2m-1} a_s e^{sx} \right) \\
G - \sum_{t=0}^{2m} a_t e^{tx} &= e^x p^1_{2^m-1:2^m-1} G - e^x p^1_{2^m-1:2^m-1} \sum_{s=0}^{2m-1} a_s e^{sx} \\
G \left( e^x p^1_{2^m-1:2^m-1} - 1 \right) &= e^x p^1_{2^m-1:2^m-1} \sum_{s=0}^{2m} a_s e^{sx} - \sum_{t=0}^{2m} a_t e^{tx} - e^x p^1_{2^m-1:2^m-1} a_{2m} e^{2mx} \\
G &= \frac{\left( e^x p^1_{2^m-1:2^m-1} - 1 \right) \left( \sum_{s=0}^{2m} a_s e^{sx} \right) - e^x p^1_{2^m-1:2^m-1} a_{2m} e^{2mx}}{p^1_{2^m-1:2^m-1} e^x - 1} \\
G &= \sum_{s=0}^{2m} a_s e^{sx} + \frac{p^1_{2^m-1:2^m-1} a_{2m} e^{(2m+1)x}}{1 - p^1_{2^m-1:2^m-1} e^x}
\end{aligned}
$$

where,

$$
\begin{aligned}
a_1 &= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) p^1_{2i:j} \\
a_2 &= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \left\{ p^1_{2i\,:\,2^{m-1}+\frac{1}{2}[j-(j \bmod 2)]} \right\} \times \left\{ p^1_{1+4[i \bmod 2^{m-2}]\,:\,j} \right\} \\
&\vdots \\
a_{2m} &= \sum_{i=0}^{2^{m-1}-1} \sum_{j=0}^{2^{m-1}-1} \pi(i\_j) \\
&\qquad \times \prod_{k=0}^{2m-1} \left\{ p^1_{2^{\min(k,m)}-1+\left[2^{k+1}(i \bmod 2^{m-k-1})\right]\,:} \right. \\
&\qquad\qquad \left. \sum_{t=0}^{\min(2m-k-2,m-1)} \{2^{m-t-1}\} + \frac{1}{2^{2m-k-1}} \left[ j-(j \bmod 2^{2m-k-1}) \right] \right\}
\end{aligned}
$$

□

## B.6 Expected Run Length for $m = 1$ Obtained from Moment Generating Function

$$E\left[\textbf{run length}\right] = p^1_{0:0} + \frac{p^1_{0:1}p^1_{1:0}\left(2 - p^1_{1:1}\right)}{\left(1 - p^1_{1:1}\right)^2} \qquad \text{for } |p^1_{1:1}|$$

*Proof.*

$$G(x) = \frac{\left(p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1}\right)e^{2x} + p^1_{0:0}e^x}{1 - p^1_{1:1}e^x}$$

$$G'(x) = \frac{\left[2\left(p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1}\right)e^{2x} + p^1_{0:0}e^x\right]\left(1 - p^1_{1:1}e^x\right)}{\left(1 - p^1_{1:1}e^x\right)^2} - \frac{\left[\left(p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1}\right)e^{2x} + p^1_{0:0}e^x\right]\left(p^1_{1:1}e^x\right)}{\left(1 - p^1_{1:1}e^x\right)^2}$$

$$\begin{aligned}
G'(0) &= \frac{\left[2\left(p^1_{0:1}p^1_{1:} - p^1_{0:0}p^1_{1:1}\right) + p^1_{0:0}\right]\left(1 - p^1_{1:1}\right) - \left[\left(p^1_{0:1}p^1_{1:0} - p^1_{0:0}p^1_{1:1}\right) + p^1_{0:0}\right]\left(p^1_{1:1}\right)}{\left(1 - p^1_{1:1}\right)^2} \\
&= \frac{2p^1_{0:1}p^1_{1:0} - 2p^1_{0:0}p^1_{1:1} + p^1_{0:0} - p^1_{0:1}p^1_{1:0}p^1_{1:1} + p^1_{0:0}\left(p^1_{1:1}\right)^2}{\left(1 - p^1_{1:1}\right)^2} \\
&= \frac{p^1_{0:0}\left[\left(p^1_{1:1}\right)^2 - 2p^1_{1:1} + 1\right]}{\left(1 - p^1_{1:1}\right)^2} + \frac{2p^1_{0:1}p^1_{1:0} - p^1_{0:1}p^1_{1:0}p^1_{1:1}}{\left(1 - p^1_{1:1}\right)^2} \\
&= p^1_{0:0} + \frac{2p^1_{0:1}p^1_{1:0} - p^1_{0:1}p^1_{1:0}p^1_{1:1}}{\left(1 - p^1_{1:1}\right)^2} \\
&\text{since } \left(1 - p^1_{1:1}\right)^2 = \left(p^1_{1:1}\right)^2 - 2p^1_{1:1} + 1 \\
&= p^1_{0:0} + \frac{p^1_{0:1}p^1_{1:0}\left(2 - p^1_{1:1}\right)}{\left(1 - p^1_{1:1}\right)^2} \\
&= E\left[\textbf{run length}\right] \text{ from Theorem 6.3.}
\end{aligned}$$

□

## B.7 Theorem 6.12 (Marginal Likelihood, $m \geq 1$)

$$\begin{aligned}\mathcal{L} &= \prod_{i=0}^{2^{2m}-1} \pi\left(\frac{1}{2^m}\left[i-(i \bmod 2^m)\right] \,:\, i \bmod 2^m\right)^{n_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}} \\ &= \prod_{i=0}^{2^{2m}-1} \pi(d(i))^{n_{d(i)}} \\ &= \prod_{i=0}^{2^{2m}-2} \pi(d(i))^{n_{d(i)}}\left[1-\left(\sum_{j=0}^{2^{2m}-2}\pi(d(j))\right)\right]^{N-\left(\sum_{j=0}^{2^{2m}-2} n_{d(j)}\right)},\end{aligned}$$

where, $d(i) = \frac{1}{2^m}\left[i-(i \bmod 2^m)\right] \,:\, i \bmod 2^m$

*Proof.* Given a sequence of letters 0 and 1 of length $N+2m$, the sequence can be defined by its associated de Bruijn words of length $m \geq 1$. $\pi(i:j)$ gives the probability of obtaining the word $i:j$ and $N$ gives the total number of words. The joint distribution is given by the product of the respective $\pi$'s for each word and like terms are collected. Since each word consists of $2m$ letters, there are $2^{2m}$ possible words. Each non-directional word is of the form $w_1 : w_2$, where each of these sequences, $w_k$, is of length $m$. All possible words must consist of all combinations of $w_1$ with $w_2$. Hence to give all possible words, if one $w_k$ repeats each word $2^m$ times in turn and the other $w_k$ repeats the whole list $2^m$ times this will produce all combinations. $\frac{1}{2^m}\left[i-(i \bmod 2^m)\right]$ for $i = 1, 2, ...$ gives the sequence $0, ..., 0, 1, ..., 1, ..., 2^m, ..., 2^m$ whilst $i \bmod 2^m$ gives the list $1, 2, ..., 2^m$, $2^m$ times. Hence this gives:

$$\begin{aligned}\mathcal{L} &= \prod_{i=0}^{2^{2m}-1} \pi\left(\frac{1}{2^m}\left[i-(i \bmod 2^m)\right] \,:\, i \bmod 2^m\right)^{n_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}} \\ &= \prod_{i=0}^{2^{2m}-1} \pi(d(i))^{n_{d(i)}},\end{aligned}$$

where $d(i) = \frac{1}{2^m}\left[i-(i \bmod 2^m)\right] \,:\, i \bmod 2^m$.

Since the sum of all $n_i$ gives the total number of words, $N$, and the sum of the marginal probabilities of the words equals one, this gives the following:

$$\mathcal{L} = \prod_{i=0}^{2^{2m}-2} \pi(d(i))^{n_{d(i)}}\left[1-\left(\sum_{j=0}^{2^{2m}-2}\pi(d(j))\right)\right]^{N-\left(\sum_{j=0}^{2^{2m}-2} n_{d(j)}\right)}.$$

□

## B.8 Theorem 6.14 (Conditional Word Likelihood, $m \geq 1$)

$$\begin{aligned}\mathcal{L} &= \prod_{i=0}^{2^{2m}-1} \left(p^0_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}\right)^{n^0_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}} \\ &\quad \times \left(p^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}\right)^{n^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}} \\ &= \prod_{i=0}^{2^{2m}-1} \left(1-p^1_{d(i)}\right)^{n^0_{d(i)}} \left(p^1_{d(i)}\right)^{n^1_{d(i)}},\end{aligned}$$

where, $d(i) = \frac{1}{2^m}\left[i - (i \bmod 2^m)\right] \,:\, i \bmod 2^m$

*Proof.* Assume a sequence of letters, $L = l_1, l_2, ..., l_n$, where $l_i \in [0, 1]$ and the ordering is fixed. This sequence can be expressed in terms of its de Bruijn words such that $L = w_{l_1:l_3}, w_{l_2:l_4}, ..., w_{l_{n-2}:l_n}$, where $w_{l_i:l_{i+2}}$ is the de Bruijn word of length $m$. Consider the joint distribution of this sequence. Consider the joint distribution of this sequence. Starting from $w_{l_1:l_3}$, $l_2$ can either be a 1 or 0 with probabilities $p^1_{w_{l_1:l_3}}$ and $p^0_{w_{l_1:l_3}} = 1 - p^1_{w_{l_1:l_3}}$. Then $l_3$ can either be a 1 or 0 with probabilities $p^1_{w_{l_2:l_4}}$ and $p^0_{w_{l_2:l_4}} = 1 - p^1_{w_{l_2:l_4}}$. This continues for all letters in $L$, where $l_{n-1}$ is either a 0 or a 1 with probabilities $p^1_{w_{l_{n-2}:l_n}}$ and $p^0_{w_{l_{n-2}:l_n}} = 1 - p^1_{w_{l_{n-2}:l_n}}$. This produces the joint distribution, $\mathcal{L}(L|p) = p^{l_2}_{w_{l_1:l_3}} \times p^{l_3}_{w_{l_2:l_4}} \times ... \times p^{l_{n-1}}_{w_{l_{n-2}:l_n}}$. Like terms are collected for each possible transition probability.

Since each word consists of $2m$ letters, there are $2^{2m}$ possible words. Therefore there are $2^{2m+1}$ possible transition probabilities since each possible word can either transition to a 0 or a 1. Each non-directional word is of the form $w_1 : w_2$, where each of these sequences, $w_k$, is of length $m$. All possible words must consist of all combinations of $w_1$ with $w_2$. Hence to give all possible words, if one $w_k$ repeats each word $2^m$ times in turn and the other $w_k$ repeats the whole list $2^m$ times this will produce all combinations. $\frac{1}{2^m}\left[i - (i \bmod 2^m)\right]$ for $i = 1, 2, ...$ gives the sequence $0, ..., 0, 1, ..., 1, ..., 2^m, ..., 2^m$ whilst $i \bmod 2^m$ gives the list $1, 2, ..., 2^m$, $2^m$ times.

Each word can then transition to either a 0 or a 1 which gives the following result:

$$\begin{aligned}\mathcal{L} &= \prod_{i=0}^{2^{2m}-1} \left(p^0_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}\right)^{n^0_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}} \\ &\qquad \times \left(p^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}\right)^{n^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}} \\ &= \prod_{i=0}^{2^{2m}-1} \left(1-p^1_{d(i)}\right)^{n^0_{d(i)}} \left(p^1_{d(i)}\right)^{n^1_{d(i)}},\end{aligned}$$

where, $d(i) = \frac{1}{2^m}\left[i - (i \bmod 2^m)\right] \,:\, i \bmod 2^m$ □

## B.9 Theorem 6.15 (Posterior Distribution for de Bruijn Conditional Word Probabilities, $m \geq 1$)

$$\begin{aligned} P(p|seq) &= \frac{\mathcal{L}(seq|p,m)P(p|m)}{P(seq)} \\ &= \frac{\mathcal{L}(seq|p,m)P(p|m)}{\int \mathcal{L}(seq|p,m)P(p|m)dp}, \end{aligned}$$

where,

$$\begin{aligned} \mathcal{L}(seq|p,m)P(p|m) &= \prod_{i=0}^{2^{2m}-1} \left(1 - p^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}\right)^{n^0_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}+\beta_{i+1}-1} \\ &\quad \times \left(p^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}\right)^{n^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}+\alpha_{i+1}-1} \\ &= \prod_{i=0}^{2^{2m}-1} \left(1 - p^1_{d(i)}\right)^{n^0_{d(i)}+\beta_{i+1}-1} \left(p^1_{d(i)}\right)^{n^1_{d(i)}+\alpha_{i+1}-1}, \end{aligned}$$

and

$$\begin{aligned} &\int P(seq|p,m)P(p|m)dp \\ &= \prod_{i=0}^{2^{2m}-1} \frac{\Gamma\left(n^0_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m} + \beta_{i+1}\right)\Gamma\left(n^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m}) + \alpha_{i+1}\right)}{\Gamma\left(n^0_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m} + n^1_{\frac{1}{2^m}[i-(i \bmod 2^m)] \,:\, i \bmod 2^m} + \beta_{i+1} + \alpha_{i+1}\right)} \\ &= \prod_{i=0}^{2^{2m}-1} \frac{\Gamma\left(n^0_{d(i)} + \beta_{i+1}\right)\Gamma\left(n^1_{d(i)}) + \alpha_{i+1}\right)}{\Gamma\left(n^0_{d(i)} + n^1_{d(i)} + \beta_{i+1} + \alpha_{i+1}\right)}, \end{aligned}$$

with $d(i) = \frac{1}{2^m}\left[i - (i \bmod 2^m)\right] \,:\, i \bmod 2^m$

*Proof.* The equation:

$$\begin{aligned} P(p|seq) &= \frac{\mathcal{L}(seq|p,m)P(p|m)}{P(seq)} \\ &= \frac{\mathcal{L}(seq|p,m)P(p|m)}{\int \mathcal{L}(seq|p,m)P(p|m)dp}, \end{aligned}$$

is taken from Bayes' theorem where the likelihood is given in Theorem 6.14 and the prior is defined to be a product of beta densities:

$$P(p|m) = \prod_{i=0}^{2^m-1} \frac{\Gamma(\alpha_{i+1}+\beta_{i+1})}{\Gamma(\alpha_{i+1})\Gamma(\beta_{i+1})} p^{\alpha_{i+1}-1}(1-p)^{\beta_{i+1}-1},$$

for unknown parameters $\alpha$ and $\beta$. Substituting this in gives:

$$\begin{aligned} P(p|seq) &\propto \mathcal{L}(seq|p,m)P(p|m) \\ &= \prod_{i=0}^{2^{2m}-1} \left[ \left(1-p^1_{d(i)}\right)^{n^0_{d(i)}} \left(p^1_{d(i)}\right)^{n^1_{d(i)}} \right. \\ &\qquad \left. \times \left(1-p^1_{d(i)}\right)^{\beta_{i+1}-1} \left(p^1_{d(i)}\right)^{\alpha_{i+1}-1} \right] \\ &= \prod_{i=0}^{2^{2m}-1} \left(1-p^1_{d(i)}\right)^{n^0_{d(i)}+\beta_{i+1}-1} \left(p^1_{d(i)}\right)^{n^1_{d(i)}+\alpha_{i+1}-1}, \end{aligned}$$

where, $d(i) = \frac{1}{2^m}\left[i - (i \bmod 2^m)\right] \ : \ i \bmod 2^m$.

Both the prior, $P(p|m)$, and the posterior, $P(p|seq)$, take the form of a product of beta densities, hence there is a conjugate relationship and the following is true:

$$\int P(seq|p,m)P(p|m)dp = \prod_{i=0}^{2^{2m}-1} \frac{\Gamma\left(n^0_{d(i)}+\beta_{i+1}\right)\Gamma\left(n^1_{d(i)})+\alpha_{i+1}\right)}{\Gamma\left(n^0_{d(i)}+n^1_{d(i)}+\beta_{i+1}+\alpha_{i+1}\right)}.$$

□